ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ
ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ
ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Η γλώσσα προγραμματισμού ανοιχτού λογισμικού Processing

Μπουτοβίνας Αντώνιος

Επιβλέπων καθηγητής: **Ράπτης Πασχάλης**

Θεσσαλονίκη, Μάιος 2013

Πίνακας περιεχομένων

Περίληψη

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι η παρουσίαση της γλώσσας προγραμματισμού ανοικτού κώδικα Processing . Η Processing χρησιμοποιείτε για την δημιουργία εικόνων ,γραφικών δύο ή τριών διαστάσεων (2D ,3D graphics),κινούμενα σχέδια και αλληλεπιδράσεις .Παρόλο που η γλώσσα Processing αρχικά αναπτύχθηκε για να για να διδάξει τις βασικές αρχές του προγραμματισμού μέσα από ένα οπτικό πλαίσιο έχει εξελίχθη σε ένα επαγγελματικό εργαλείο .Πλέον υπάρχουν πάνω από 100 εξωτερικές βιβλιοθήκες που μπορούν να συνδεθούν με την γλώσσα Processing επεκτείνοντας τις δυνατότητες της γλώσσας .Οι βιβλιοθήκες της Processing καλύπτουν ένα ευρύ φάσμα λειτουργιών. Οι ποιο διαδεδομένες βιβλιοθήκες αφορούν τα γραφικά τριών διαστάσεων (πχ βιβλιοθήκη OpenGL) , την αναπαραγωγή ήχου την εγγραφή ήχου σε αρχείο ,την διαχείριση των εισόδων και τον εξόδων του ήχου (βιβλιοθήκη Minim) ,την αναπαραγωγή αρχείων βίντεο ,την αναπαραγωγή εικόνας απευθείας από μια κάμερα, την εγγραφή εικόνας σε ένα αρχείο (βιβλιοθήκη video) , την επικοινωνία με άλλες συσκευές εισόδου και εξόδου(serial, touchatag). Βέβαια υπάρχουν και άλλες βιβλιοθήκες που καλύπτουν τομείς πιο περίπλοκους όπως είναι οι βιβλιοθήκες μαθηματικών εξισώσεων και προσομοιώσεων (Physics, MatrixMath, Cell Noise),οι βιβλιοθήκες των animation (Ani, gifAnimation) και βιβλιοθήκες που διαχειρίζονται δεδομένα και πρωτόκολλα όπως είναι οι xmlrpclib , proHTML , proXML και Arduino Library.

1 Εισαγωγή

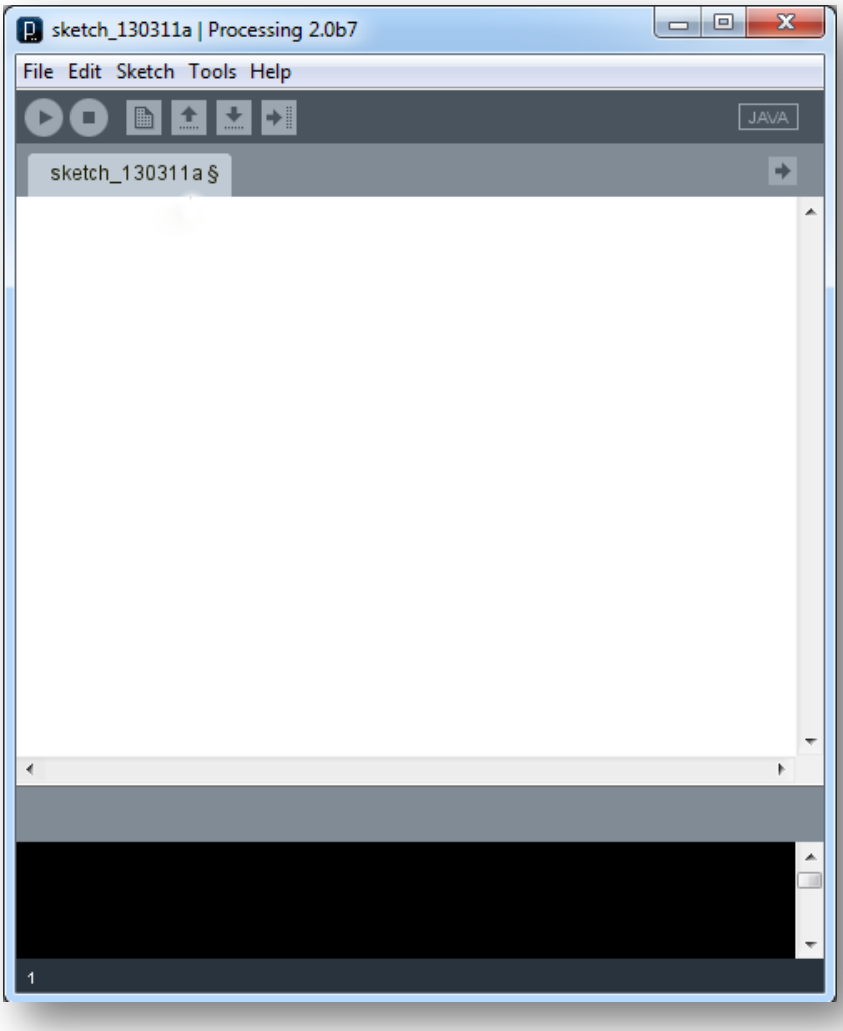
Η Processing δημιουργήθηκε το 2001 από τους Casey Reas και Benjamin Fry . Η Processing βασίζεται στη γλώσσα Java, αλλά χρησιμοποιεί μια πιο απλοποιημένη σύνταξη. Οι βασικοί λόγοι που τους ώθησαν να δημιουργήσουν αυτήν την νέα γλώσσα προγραμματισμού ήταν δύο. Πρώτον ήθελαν να δημιουργήσουν έναν πρότυπο για τον τύπο του λογισμικού που ανέπτυσσαν συνήθως που ήταν κυρίως διαδραστικές εφαρμογές .Αυτό θα τους επέτρεπε να δοκιμάζουν εύκολα και γρήγορα τις ιδέες τους σε κώδικα σε σύγκριση με την δόκιμη των ιδεών τους σε κάποια άλλη γλώσσα προγραμματισμού όπως για παράδειγμα η C++.Ο δεύτερος λόγος ήταν η δημιουργία μιας γλώσσας προγραμματισμού για διδακτικούς λόγους . Με την ανάπτυξη της Processing θα μπορούσαν οι σπουδαστές των καλών τεχνών να διδαχτούν εύκολα προγραμματισμό και οι σπουδαστές μιας σχολής πληροφορικής θα είχαν στην διάθεση τους έναν ποίο εύκολο τρόπο για να δημιουργούν γραφικά και διαδραστικές εφαρμογές.

Η αρχική ανάπτυξη της Processing κράτησε αρκετό καιρό .Ο κύκλος ζωής της έκδοσης της γλώσσας ανοικτού λογισμικού Processing είναι εν συντομία , η Processing υπήρχε στην Alpha έκδοση της από τον Αύγουστο του 2002 μέχρι και τον Απρίλιο του 2005 έπειτα στην Beta έκδοση της μέχρι τον Νοέμβριο του 2008 . Κατά τη διάρκεια αυτής της περιόδου, η Processing χρησιμοποιήθηκε για διδακτικούς σκοπούς αλλά και γενικά από χιλιάδες χρήστες σε όλο τον κόσμο . Επίσης η γλώσσα και το περιβάλλον του λογισμικού, επανεξετάζονταν συνεχώς. Στις 29 Νοεμβρίου 2008,δρομολογήθηκε η έκδοση της Processing 1.0.Η ανάπτυξη της γλώσσας συνεχίζεται διαρκώς έχοντας ως αποτέλεσμα της παραγωγή νέων εκδόσεων της Processing με πιο πρόσφατη στην Processing 2.0 8beta.

Τα μαθήματα των περισσότερων γλωσσών προγραμματισμού επικεντρώνονται κυρίως στην δομή της γλώσσας και στην θεωρία .Για να έρθει σε επαφή κάποιος με διεπαφές, γραφικά,animation η οτιδήποτε άλλο πρέπει πρώτα να περάσει πολλές εβδομάδες μελετώντας αλγορίθμους και μεθόδους .Έχει παρατηρηθεί όλα αυτά τα χρόνια ότι πολλοί άνθρωποι που ενδιαφέρθηκαν να παρακολουθήσουν κάποια μαθήματα προγραμματισμού γρήγορα έχασαν τον ενδιαφέρον τους γιατί έπρεπε να ακολουθήσουν την μέθοδο που αναφέρεται παραπάνω ,δηλαδή να σπαταλήσουν αρκετό χρόνο στην εκμάθηση της γλώσσας μέχρι να φτάσουν στο κομμάτι της δημιουργίας .Η Processing προσφέρει έναν εναλλακτικό τρόπο ώστε να διδαχτεί κάποιος προγραμματισμό πιο εύκολα μέσα από την δημιουργία διαδραστικών γραφικών .Οι μαθητές ενθαρρύνονται από τα άμεσα οπτικά αποτελέσματα που παίρνουν και βρίσκουν κίνητρα να ασχοληθούν με την δημιουργία εικόνων ,γραφικών και διαδραστικών διεπαφών.

Αν θέλουμε σύντομα να δούμε από τι από αποτελείται η Processing θα καταλήξουμε στα εξής :

- Το Περιβάλλον Ανάπτυξης της Processing (Processing Development Environment , PDE). Είναι το λογισμικό που τρέχει όταν κάνουμε διπλό κλικ στο εικονίδιο Processing. Το PDE είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης με ένα μινιμαλιστικό σχεδιασμό του συνόλου των χαρακτηριστικών της Processing που έχει ως σκοπό την εισαγωγή στον προγραμματισμό ή την δοκιμή ιδεών .Μπορείτε να κατεβάσετε το PDE από το http://processing.org/download/
- Ένα σύνολο από συναρτήσεις και μεθόδους που αποτελούν τον πυρήνα του API της Processing, καθώς και αρκετές βιβλιοθήκες που υποστηρίζουν πιο προηγμένα χαρακτηριστικά, όπως η δημιουργία γραφικών με OpenGL, ανάγνωση αρχείων XML, καθώς και την αποθήκευση πολύπλοκων γραφικών σε μορφή PDF.
- Ένα συντακτικό πανομοιότυπο με αυτό της Java,και μερικά επιπρόσθετα χαρακτηριστικά που αφορούν τα γραφικά και τις αλληλεπιδράσεις .
- Μια ενεργή διαδικτυακή κοινότητα στο http://processing.org



2 Σχεδίαση

Αρχικά η οθόνη του υπολογιστή μας είναι ένα πλέγμα από φωτεινά στοιχεία που ονομάζονται pixels .Κάθε pixel έχει μια συγκεκρίμένη θέση μέσα σε αυτό το πλέγμα που προσδιορίζεται από τις συντεταγμένες. Στην Processing η συντεταγμένη x ορίζει την απόσταση από το αριστερό άκρο του παραθύρου ενώ η συντεταγμένη y ορίζει την απόσταση από το άνω άκρο του παραθύρου .Γράφουμε τις συντεταγμένες μας με αυτήν την μορφή (x, y).Οπότε αν έχουμε ένα παράθυρο που είναι 200x200 pixels το πάνω αριστερό άκρο είναι το (0,0) το κέντρο του παραθύρου είναι το (100,100) και το και το κάτω δεξί άκρο είναι το (199,199).

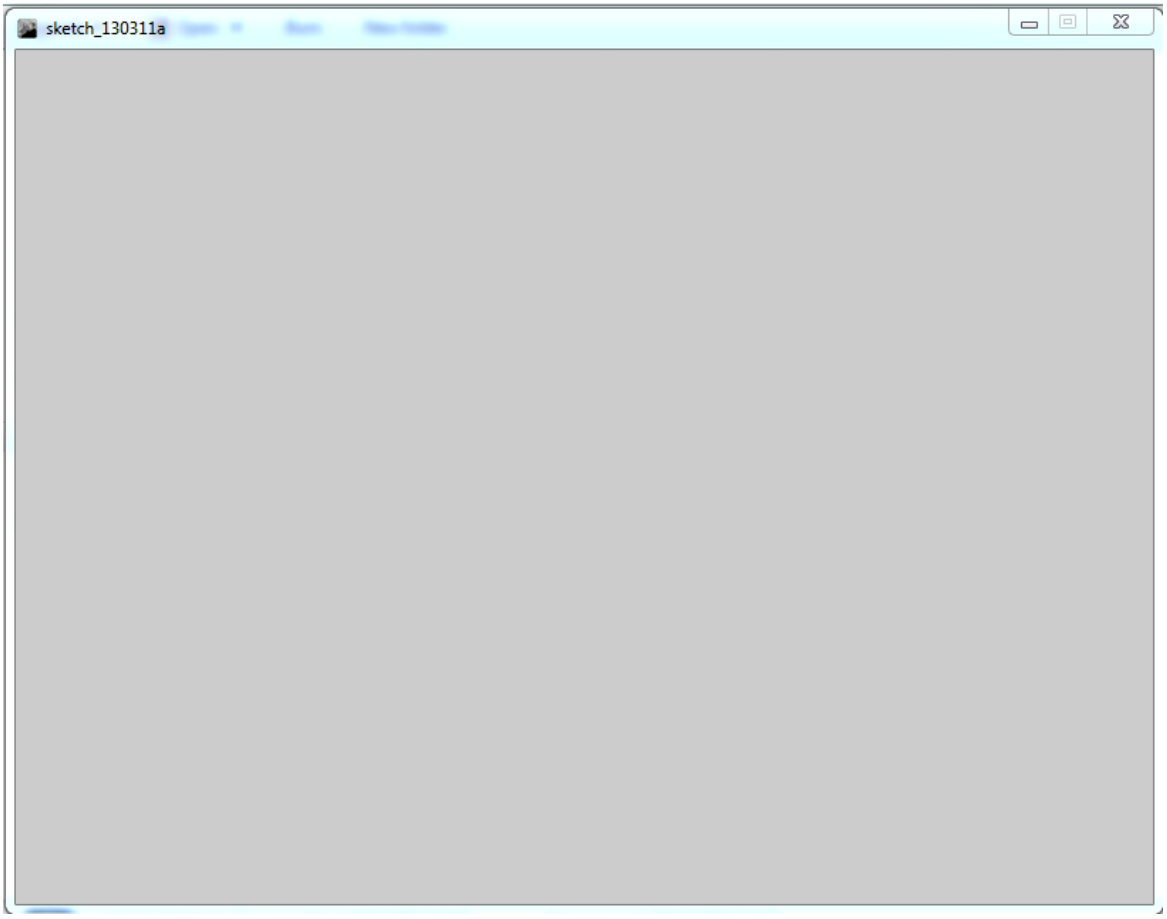
Το παράθυρο δημιουργείτε και μέσα σε αυτό σχεδιάζονται οι εικόνες και τα γραφικά με την βοήθεια των συναρτήσεων .Οι συναρτήσεις έχουν παραμέτρους οι οποίες ορίζουν το αποτέλεσμα τους. Για παράδειγμα, σχεδόν όλα τα πρόγραμμα της Processing έχουν την συνάρτηση size() η οποία ορίζει το πλάτος και το ύψος του παραθύρου σε περίπτωση που δεν οριστεί η size() το μέγεθος του παραθύρου είναι 100X100 pixels.

2.1 Σχεδίαση παραθύρου

Η συνάρτηση size() έχει δύο παραμέτρους η πρώτη ορίζει το πλάτος του παραθύρου ενώ η δεύτερη ορίζει το ύψος του . Αν θέλουμε να σχεδιάσουμε ένα παράθυρο με διαστάσεις 800X600 πρέπει να γράψουμε την παρακάτω γραμμή κώδικα

size(800, 600);

Αν εκτελέσουμε τον παραπάνω κώδικα θα πάρουμε ως αποτέλεσμα το παράθυρο με διαστάσεις 800X600.

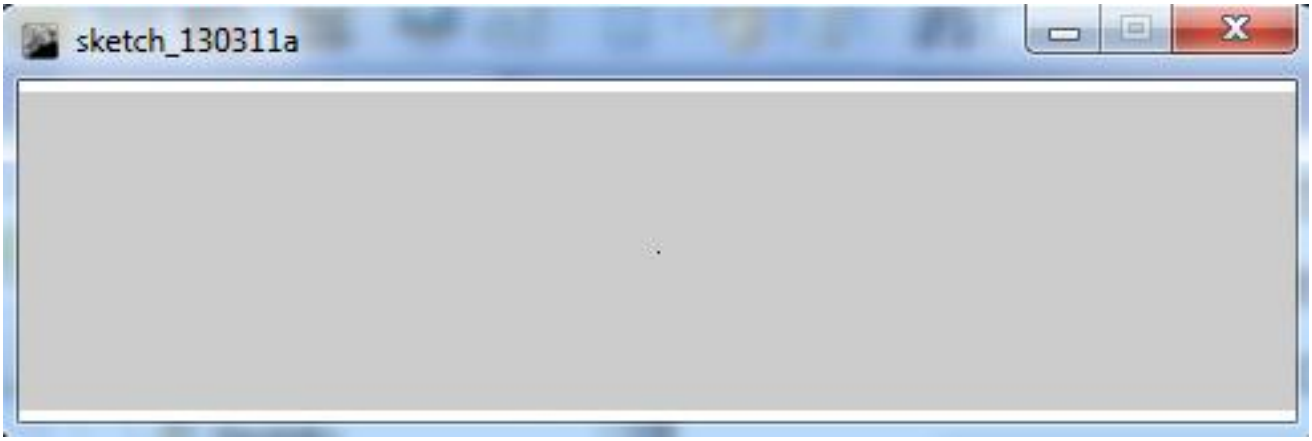


2.2 Σχεδίαση ενός σημείου

Για να χρωματίσουμε ένα pixel του παραθύρου χρησιμοποιούμε την συνάρτηση point() η οποία έχει δύο παραμέτρους. Η πρώτη ορίζει την συντεταγμένη X και η δεύτερη ορίζει την συντεταγμένη Y του pixel που θέλουμε να χρωματίσουμε. Με τον παρακάτω κώδικα θα σχεδιάσουμε ένα παράθυρο και στο κέντρο του ένα σημείο.

size(480, 120);

point(240, 60);



Δίνοντας ένα δεύτερο παράδειγμα σχεδίασης σημείων θα σχεδιάσουμε τέσσερα σημεία στις τέσσερις γωνίες του παραθύρου μας.

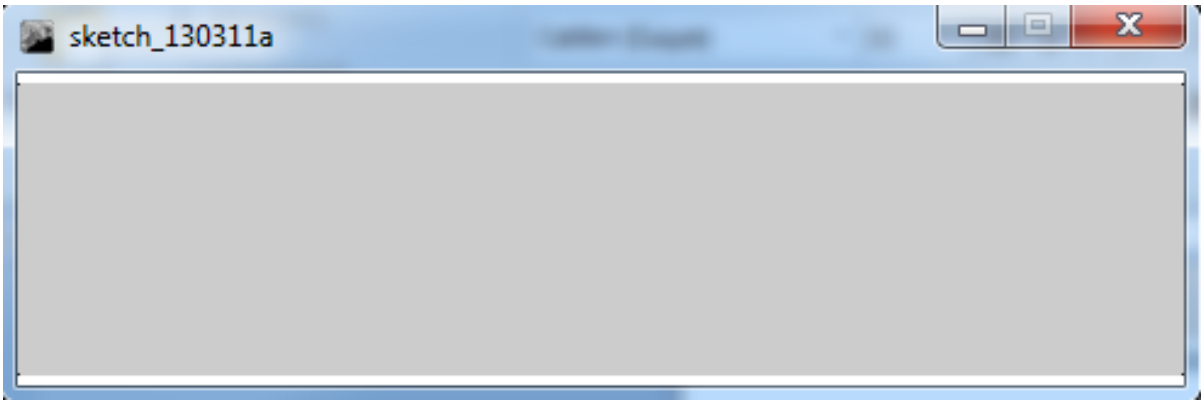
size(480, 120);

point(0, 0);

point(479, 119);

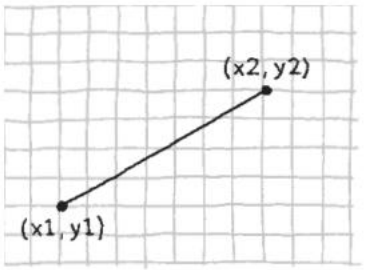
point(0,119);

point(479,0);

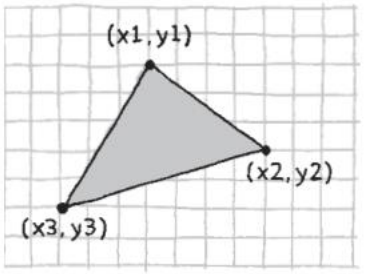


2.3 Σχεδίαση βασικών σχημάτων

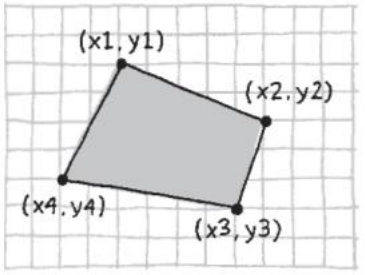
Η Processing διαθέτει ένα σύνολο συναρτήσεων των βασικών σχημάτων. Με τον συνδυασμός αυτών των βασικών σχημάτων μπορούμε να δημιουργήσουμε περιπλοκές εικόνες και γραφικά .Στην παρακάτω εικόνα μας παρουσιάζονται εν συντομία τα βασικά σχήματα της Processing που είναι η γραμμή (line),το τρίγωνο (triangle),το τετράπλευρο (quad), το ορθογώνιο παραλληλόγραμμο (rect), η έλλειψη(ellipse)και τέλος το τόξο(arc).



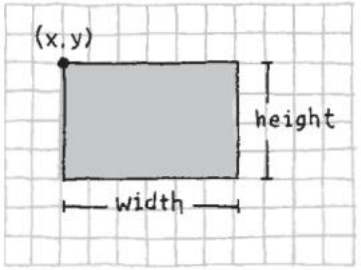
line(x1, y1, x2, y2)



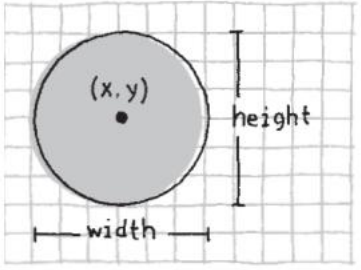
triangle(x1, y1, x2, y2, x3, y3)



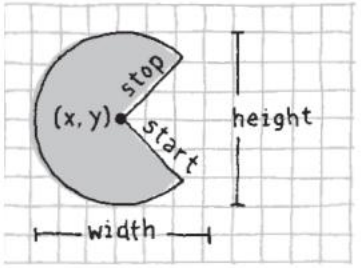
quad(x1, y1, x2, y2, x3, y3, x4, y4)



rect(x, y, width, height)



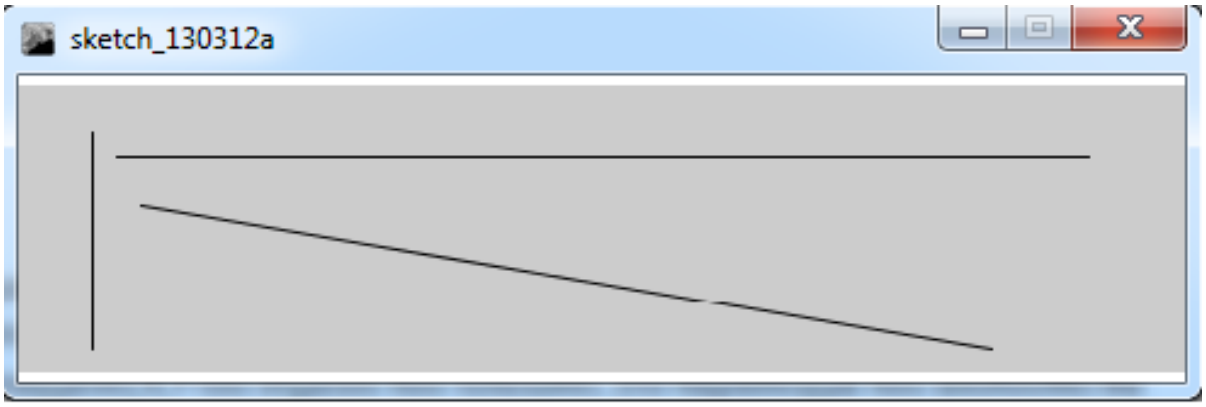
ellipse(x, y, width, height)



arc(x, y, width, height, start, stop)

2.3.1 Σχεδίαση γραμμής

Για την σχεδίαση μόνο μίας γραμμής χρειαζόμαστε την συνάρτηση line().Η συνάρτηση αυτή έχει τέσσερις παραμέτρους , οι δύο πρώτες είναι οι συντεταγμένες X,Y του σημείου που ξεκινάει η ευθεία μας και οι επόμενες δύο είναι οι συντεταγμένες X,Y του σημείου που τελειώνει. Στο παράδειγμα που ακολουθεί θα σχεδιάσουμε 3 γραμμές μια παράλληλη και μια κάθετη στον κάθετο άξονα του παραθύρου και μια τυχαία γραμμή.



size (480,120);

line(30,20,30,110);

line (40,30,440,30);

line(50,50,400,110);

2.3.2 Σχεδίαση τριγώνων

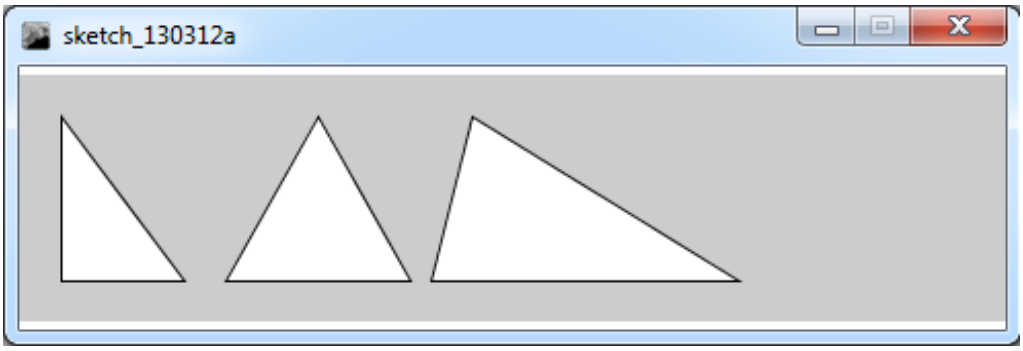
Για την σχεδίαση ενός τριγώνου χρειαζόμαστε την συνάρτηση triangle η οποία έχει έξι παραμέτρους. Οι 6 αυτές παράμετροι είναι τρία ζεύγη συντεταγμένων που ορίζουν τις τρεις κορυφές του τριγώνου .Το κομμάτι του κώδικα που ακολουθεί σχεδιάζει στην οθόνη μας τρία τρίγωνα ,ένα ορθογώνιο , ένα ισοσκελές και ένα σκαληνό.

size(480,120);

triangle(20,20,20,100,80,100);

triangle(145,20,100,100,190,100);

triangle (220,20,200,100,350,100);



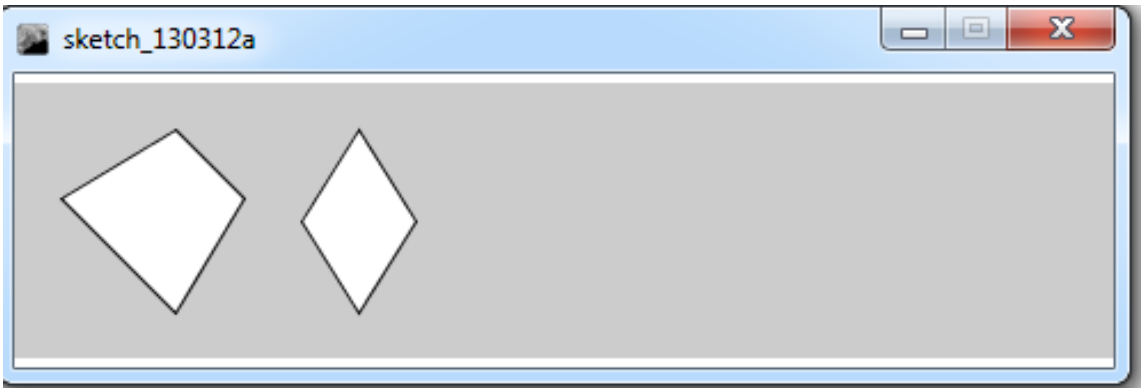
2.3.3 Σχεδίαση τετραπλεύρων

Η συνάρτηση quad μας βοηθάει για την σχεδίαση κάθε είδους τετραπλεύρου όπως για παράδειγμα ένας ρόμβος ένα τραπέζιο η ένα οποιοδήποτε τετράπλευρο .Η συγκεκριμένη συνάρτηση έχει οκτώ παραμέτρους που είναι τα τέσσερα ζεύγη των συντεταγμένων X,Y των κορυφών του τετραπλεύρου. Θα σχεδιάσουμε ως παράδειγμα ένα τυχαίο τετράπλευρο και έναν ρόμβο.

size(480,120);

quad(70,20,20,50,70,100,100,50);

quad(150,20,125,60,150,100,175,60);



2.3.4 Σχεδίαση ορθογώνιου παραλληλόγραμμου

Η συνάρτηση rect σχεδιάζει ένα ορθογώνιο παραλληλόγραμμο στην παράθυρο μας .Η συνάρτηση εξ ορισμού έχει τέσσερις παραμέτρους όπου οι πρώτες δύο μας δίνουν τις συντεταγμένες της πάνω αριστερής γωνίας ενώ η τρίτη και η τέταρτη παράμετρος αντιστοιχούν στο πλάτος και στο ύψος το σχήματος. Ο τρόπος που ερμηνεύονται οι παράμετροι μπορεί να αλλάξει με την χρήση της συνάρτησης rectMode().Αν θέλουμε το σχήμα μας να έχει στρογγυλεμένες γωνίες πρέπει να προσθέσουμε μια πέμπτη παράμετρος η οποία η τιμή της ακτίνας του τόξου και για τις τέσσερις γωνιές. Για να χρησιμοποιήσουμε διαφορετική τιμή για την ακτίνα του τόξου σε κάθε γωνία πρέπει να χρησιμοποιήσουμε συνολικά οκτώ παραμέτρους, όπου οι τελευταίες τέσσερις παράμετροι ορίζουν την τιμή του τόξου για κάθε γωνία αρχίζοντας από την πάνω αριστερή γωνία και κινούμενοι δεξιόστροφα. Άρα το συντακτικό της συνάρτησης μπορεί να έχει τις εξής μορφές

1. rect(a, b, c, d)
2. rect(a, b, c, d, r)
3. rect(a, b, c, d, tl, tr, br, bl)

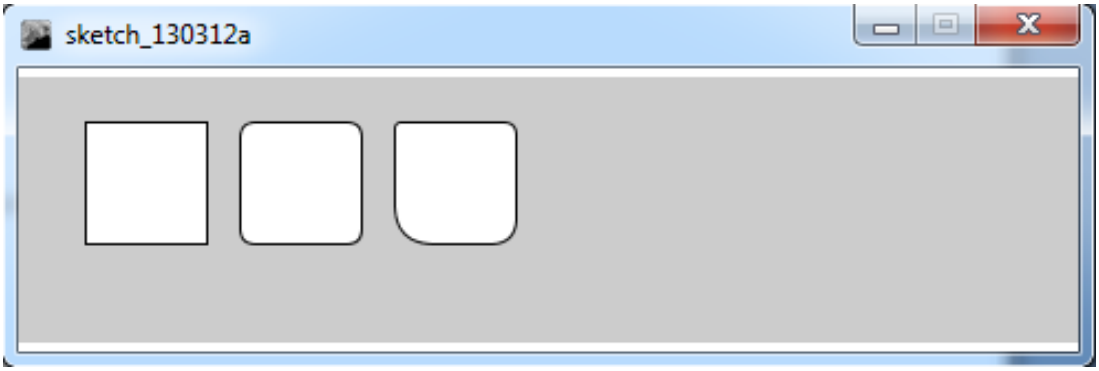
Στο παράδειγμα που ακολουθεί θα σχεδιάσουμε ένα σχήμα για κάθε τύπο του συντακτικού που αναφέρουμε παραπάνω.

size(480,120);

rect(30, 20, 55, 55);

rect(100, 20, 55, 55, 7);

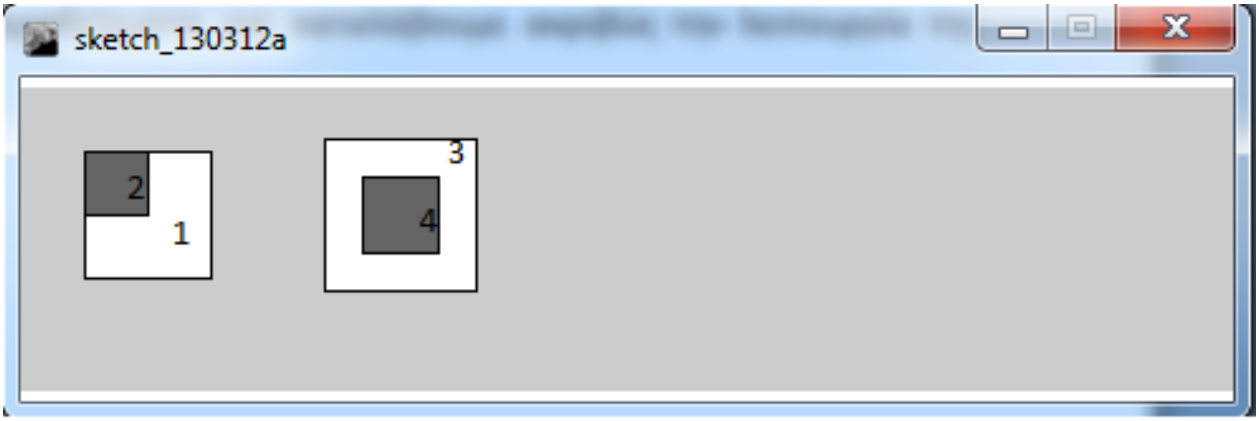
rect(170, 20, 55, 55, 3, 6, 12, 18);



2.3.4.1 Η συνάρτηση rectMode()

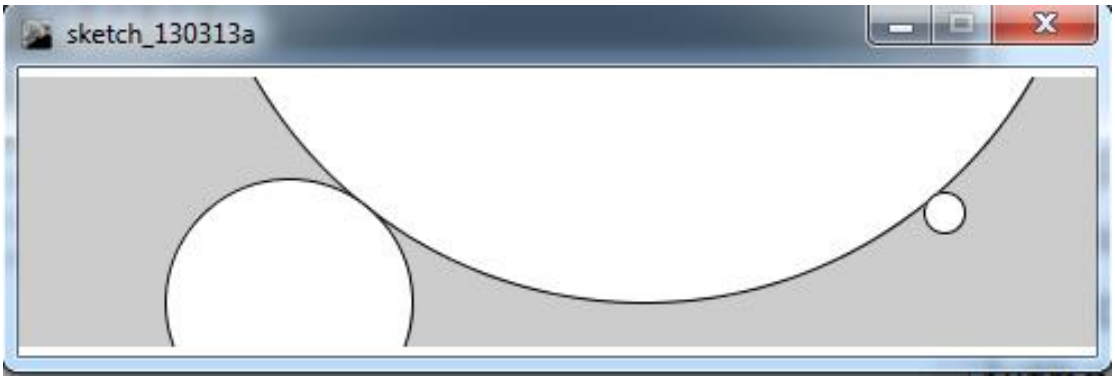
Η συνάρτηση αυτή καθορίζει με ποιον τρόπο θα σχεδιαστεί το παραλληλόγραμμο και το πως ερμηνεύονται οι παράμετροι της rect(). Η εξ ορισμού λειτουργία της συνάρτησης είναι η **rectMode(CORNER),** η οποία ερμηνεύει τις δύο πρώτες παραμέτρους της συνάρτησης rect() ως τις συντεταγμένες X,Y της πάνω αριστερής γωνίας, την τρίτη ως πλάτος του σχήματος και την τέταρτη ως ύψος του σχήματος. Η **rectMode(CORNERS)** ερμηνεύει τις πρώτες δύο παραμέτρους της rect() ως τις συντεταγμένες X,Y της πάνω αριστερής γωνίας και τις άλλες δύο παραμέτρους σαν τις συντεταγμένες της απέναντι γωνίας. Η **rectMode(CENTER)** ερμηνεύει τις πρώτες δύο παραμέτρους της rect() ως τις συντεταγμένες X,Y του κέντρου του σχήματος, την τρίτη ως πλάτος του σχήματος και την τέταρτη ως ύψος του σχήματος. Τέλος η **rectMode(RADIUS)** ερμηνεύει και αυτή τις δύο πρώτες παραμέτρους της rect() ως τις συντεταγμένες X,Y του κέντρου του σχήματος, αλλά χρησιμοποιεί την τρίτη και την τέταρτη παράμετρο για να προσδιορίσει το ήμισυ του πλάτους και του ύψους του σχήματος. Εδώ πρέπει να σημειώσουμε ότι οι παράμετροι της συνάρτησης αυτής πρέπει να είναι στα κεφαλαία για τι η Processing είναι μια case- sensitive γλώσσα. Θα δούμε κάποια παραδείγματα για καταλάβουμε ακριβώς την λειτουργία της rectMode.

```
size(480,120);

rectMode(CORNER);

rect(25, 25, 50, 50);  //σχήμα 1

rectMode(CORNERS);

fill(100); //ορίζει το χρώμα το σχήματος στο γκρι

rect(25, 25, 50, 50);  //σχήμα 2

rectMode(RADIUS);

fill(255);  //ορίζει το χρώμα το σχήματος στο λευκό

rect(150, 50, 30, 30);  //σχήμα 3

rectMode(CENTER);

fill(100);  //ορίζει το χρώμα το σχήματος στο γκρι

rect(150, 50, 30, 30);// //σχήμα 4
```



2.3.5 Σχεδίαση έλλειψης

Η συνάρτηση ellipse χρησιμοποιείται για να σχεδιάσουμε ελλείψεις στην οθόνη μας .Σε αντίθεση με την συνάρτηση rect όπου οι δύο πρώτες παράμετροι είναι οι συντεταγμένες X,Y της πάνω αριστερής γωνίας στην συνάρτηση ellipse αυτές δείχνουν το κέντρο της έλλειψης .Η τρίτη και η τέταρτη παράμετρος ορίζουν το πλάτος και το ύψος της έλλειψης. Ο τρόπος που ερμηνεύονται οι παράμετροι μπορεί να αλλάξει με την χρήση της συνάρτησης ellipseMode(). Τα αντικείμενα μπορούν να σχεδιαστούν μερικώς η εξ ολοκλήρου εκτός παραθύρου χωρίς να παράγεται κανένα σφάλμα.



size(480, 120);

ellipse(278, -100, 400, 400);

ellipse(120, 100, 110, 110);

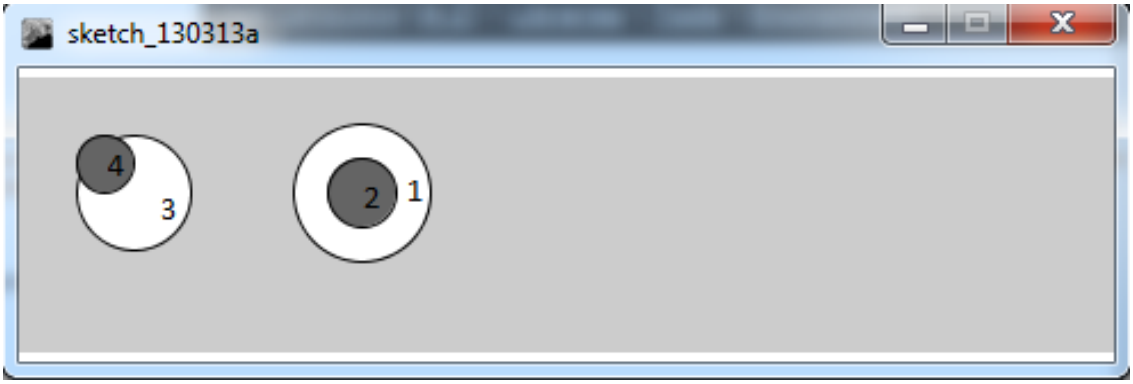
ellipse(412, 60, 18, 18);

Η Processing δεν έχει κάποια ξεχωριστή συνάρτηση για να σχεδιάζουμε κύκλους ή τετράγωνα .Αυτά τα σχήματα μπορούμε να τα σχεδιάσουμε αν χρησιμοποιήσουμε το ίδιο πλάτος και ύψος στης συναρτήσεις rect και ellipse αντίστοιχα.

2.3.5.1 Η συνάρτηση ellipseMode()

Η συνάρτηση ellipseMode τροποποιεί την θέση από την οποία ξεκινάει να σχεδιάζεται η έλλειψη . Εξ ορισμού η λειτουργία της συνάρτησης είναι η **ellipseMode(CENTER)** όπου οι δύο πρώτες παράμετροι της συνάρτησης είναι οι συντεταγμένες X,Y του κέντρου της έλλειψης , η τρίτη είναι το πλάτος του σχήματος και η τέταρτη είναι το ύψος του σχήματος. Η **ellipseMode (RADIUS)** ερμηνεύει και αυτή τις δύο πρώτες παραμέτρους της ellipse() ως τις συντεταγμένες X,Y του κέντρου του σχήματος, αλλά χρησιμοποιεί την τρίτη και την τέταρτη παράμετρο για να προσδιορίσει το ήμισυ του πλάτους και του ύψους του σχήματος. Η

ellipseMode(CORNER), η οποία ερμηνεύει τις δύο πρώτες παραμέτρους της συνάρτησης ellipse () ως τις συντεταγμένες X,Y της πάνω αριστερής γωνίας, την τρίτη ως πλάτος του σχήματος και την τέταρτη ως ύψος του σχήματος. Η **ellipseMode(CORNERS)** ερμηνεύει τις πρώτες δύο παραμέτρους της ellipse () ως τις συντεταγμένες X,Y της πάνω αριστερής γωνίας και τις άλλες δύο παραμέτρους σαν τις συντεταγμένες της απέναντι γωνίας.



size(480, 120);

ellipseMode(RADIUS);

fill(255); *//ορίζει το χρώμα το σχήματος στο λευκό*

ellipse(150, 50, 30, 30); *//σχήμα 1*

ellipseMode(CENTER);

fill(100); *//ορίζει το χρώμα το σχήματος στο γκρι*

ellipse(150, 50, 30, 30); *//σχήμα 2*

ellipseMode(CORNER);

fill(255); *//ορίζει το χρώμα το σχήματος στο λευκό*

ellipse(25, 25, 50, 50); *//σχήμα 3*

ellipseMode(CORNERS);

fill(100); *//ορίζει το χρώμα το σχήματος στο γκρι*

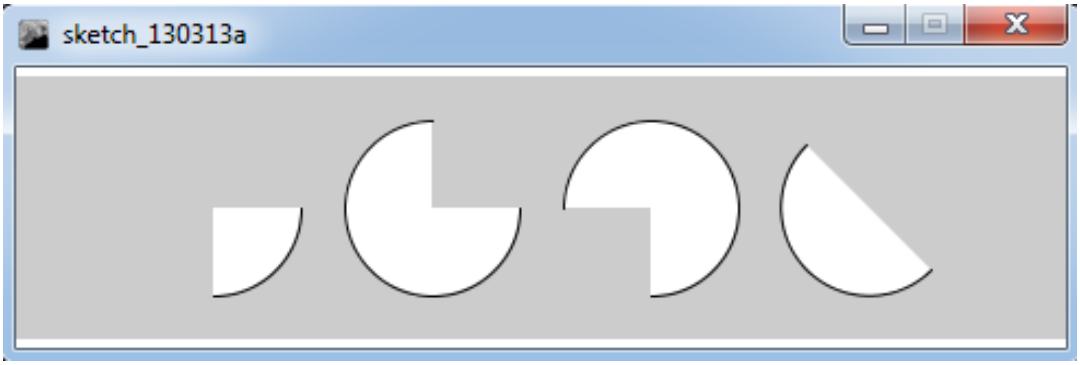
ellipse(25, 25, 50, 50); *//σχήμα 4*

2.3.6 Σχεδίαση τόξων

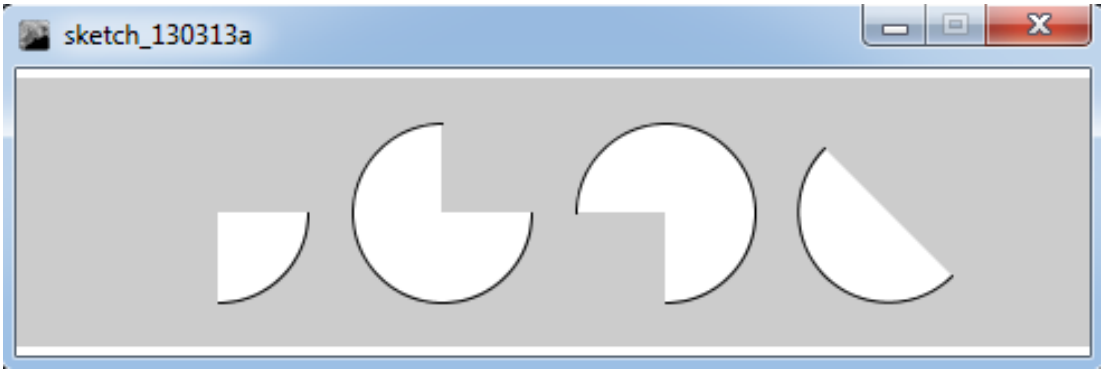
Για να σχεδιάσουμε τόξα ενός κύκλου ή μιας έλλειψης χρησιμοποιούμε την συνάρτηση arc().Στην συνάρτηση arc() οι πρώτες δύο παράμετροι προσδιορίζουν την θέση του τόξου , η τρίτη και η τέταρτη παράμετρος προσδιορίζουν το πλάτος και το ύψος του σχήματος και τέλος η πέμπτη και η έκτη παράμετρος προσδιορίζουν την γωνία που ξεκινάει και την γωνία που τελειώνει το τόξο. Αν χρησιμοποιήσουμε την συνάρτηση ellipseMode οι πρώτες τέσσερις παράμετροι λειτουργούν όπως εξηγήσαμε παραπάνω. Οι γωνίες μπορούν να δοθούν στην συνάρτηση είτε ως ακτίνια είτε ως μοίρες. Στην Processing χρησιμοποιούμε κατά κόρον τα ακτίνια, αυτό είχε σαν αποτέλεσμα να ενσωματωθούν στην γλώσσα οι συγκεκριμένες τιμές για

ακτίνια που είναι PI, QUARTER_PI, HALF_PI, και TWO_PI . Επίσης υπάρχουν και δύο βοηθητικές συναρτήσεις ,η radians() και η degrees () όπου κάνουν την μετατροπή από ακτίνια σε μοίρες και το αντίθετο .Στα ακόλουθα παραδείγματα κώδικα χρησιμοποιούμε ακτίνια και την συνάρτηση radians() παίρνοντας τα ίδια αποτελέσματα.

```
size(480, 120);

arc(90, 60, 80, 80, 0, HALF_PI);

arc(190, 60, 80, 80, 0, PI+HALF_PI);

arc(290, 60, 80, 80, PI, TWO_PI+HALF_PI);

arc(390, 60, 80, 80, QUARTER_PI, PI+QUARTER_PI);
```




```
size(480, 120);

arc(90, 60, 80, 80, 0, radians(90));

arc(190, 60, 80, 80, 0, radians(270));

arc(290, 60, 80, 80, radians(180), radians(450));

arc(390, 60, 80, 80, radians(45), radians(225));
```



2.4 Η διάταξη των σχεδίων

Όταν ένα πρόγραμμα εκτελείται ,αρχίζει από την πρώτη γραμμή του κώδικα και διαβάζει όλες τις γραμμές μία μία μέχρι την τελευταία. Αν θέλετε ένα οποιοδήποτε σχήμα να σχεδιαστεί πάνω από ένα άλλο πρέπει να τοποθετήσετε τον κώδικα που σχεδιάζει το δεύτερο σχήμα μετά το πρώτο.

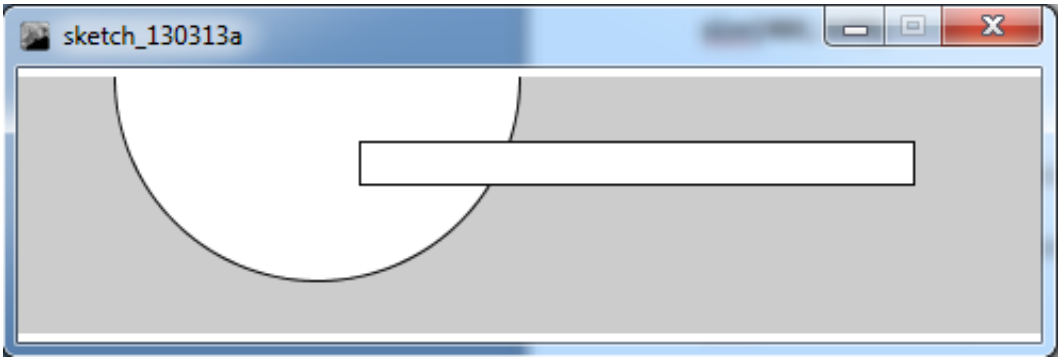
```
size(480, 120);
```

ellipse(140, 0, 190, 190);

// το παραλληλόγραμμο είναι πάνω από την έλλειψη

// γιατί είναι μετά από αυτό στον κώδικα

rect(160, 30, 260, 20);



Αν αλλάξουμε την σειρά των δύο γραμμών κώδικα θα δούμε ότι παίρνουμε διαφορετικά αποτελέσματα.

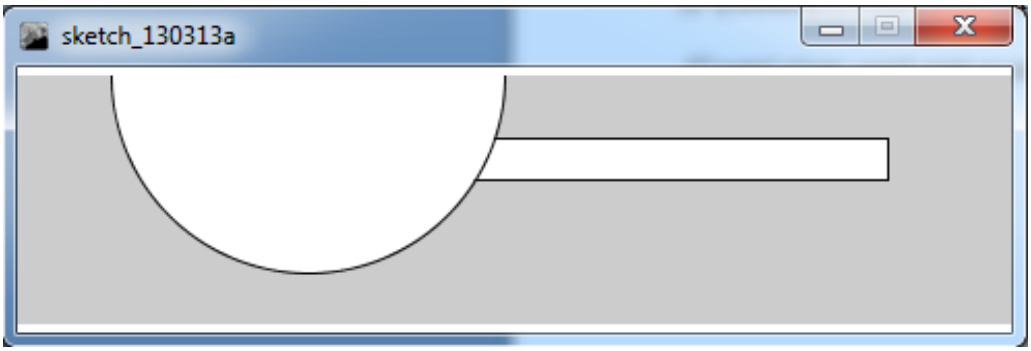
size(480, 120);

rect(160, 30, 260, 20);

// η έλλειψη είναι πάνω από το παραλληλόγραμμο

// γιατί είναι μετά από αυτό στον κώδικα

ellipse(140, 0, 190, 190);



Μπορείτε να φανταστείτε ότι είναι σαν να φτιάχνετε ένα κολάζ. Το τελευταίο κομματι που προσθέτετε είναι πάντα αυτό που είναι ορατό

2.5 Οι ιδιότητες των σχεδίων

Εδώ θα γνωρίσουμε τις πιο βασικές ιδιότητες των σχημάτων, των συναρτήσεων strokeWeight και smooth.

2.5.1 Η συνάρτηση smooth()

Η συνάρτηση smooth() μας επιτρέπει να σχεδιάζουμε τα γεωμετρικά μας σχήματα με λείες ακμές, επίσης βελτιώνει την ποιότητα των εικόνων των οποίων έχουμε αλλάξει το μέγεθος .Η συνάρτηση αυτή

είναι εξ ορισμού ενεργοποιημένη στην Processing για να την απενεργοποιήσουμε πρέπει να καλέσουμε την μέθοδο noSmooth() .

size(480, 120);

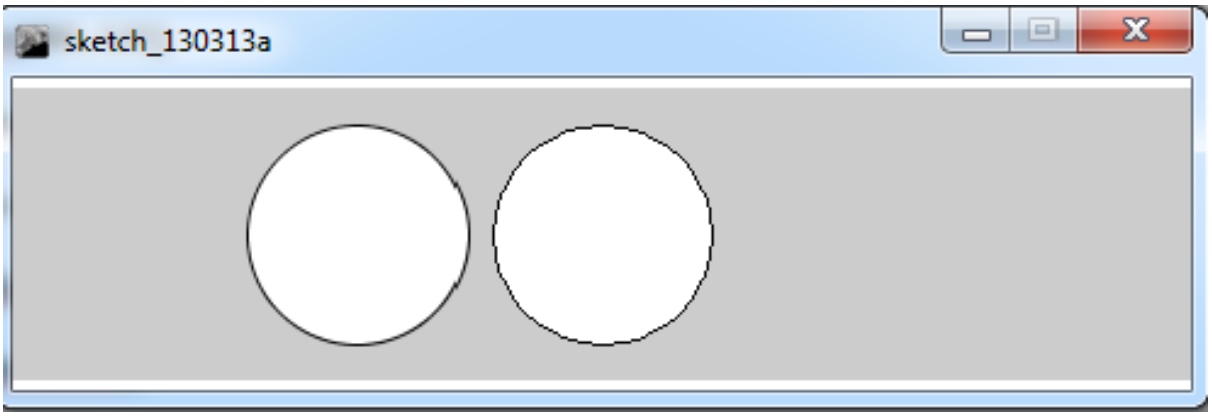
smooth(); *//θα μπορούσαμε να παραλείψουμε την κλήση της μεθόδου*

//γιατί είναι εξ ορισμού ενεργοποιημένη

ellipse(140, 60, 90, 90);

noSmooth();

ellipse(240, 60, 90, 90);



2.5.2 Η συνάρτηση strokeWeight()

Η συνάρτηση strokeWeight() ορίζει το πλάτος του περιγράμματος των διαφόρων σχημάτων. Η συνάρτηση έχει μία παράμετρο η οποία ορίζει την τιμή του πλάτους σε pixels.

size(480, 120);

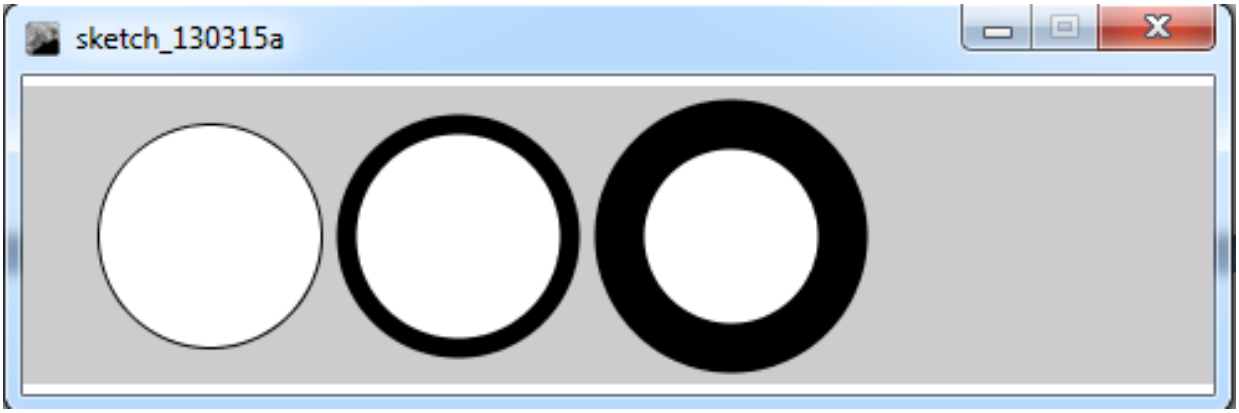
smooth();

ellipse(75, 60, 90, 90);

strokeWeight(8);

ellipse(175, 60, 90, 90);

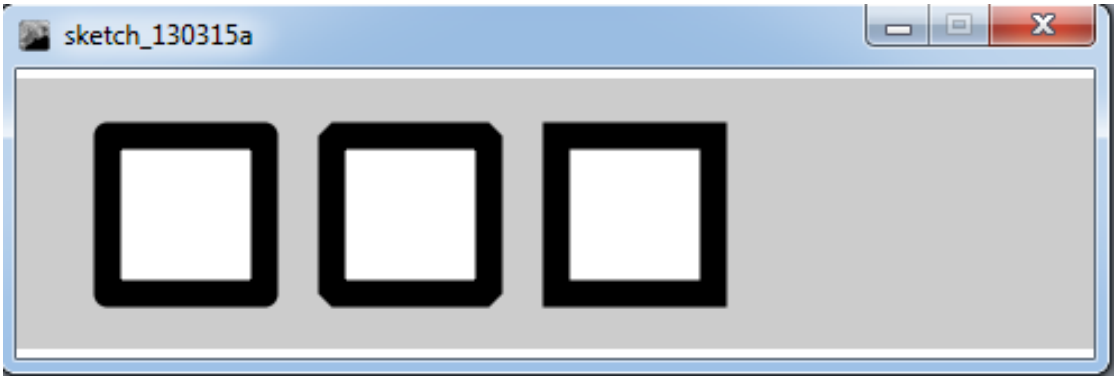
strokeWeight(20); , 60, 90, 90);



2.5.3 Οι συναρτήσεις strokeJoin() και strokeCap()

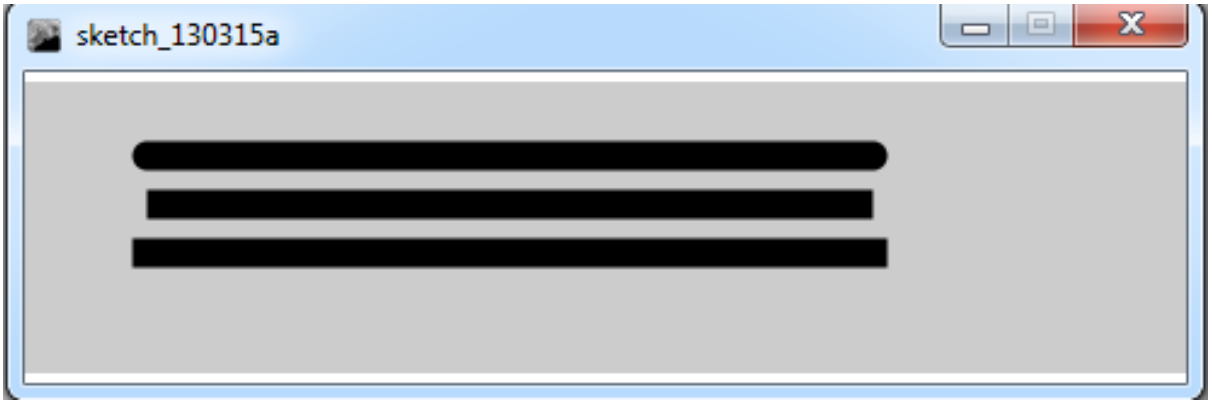
Η συνάρτηση strokeJoin() ορίζει το στιλ των αρθρώσεων των σχημάτων ,δηλαδή των γωνιών στα απλά σχήματα που έχουμε δει μέχρι στιγμής .Υπάρχουν τρεις τιμές που μπορεί να πάρει η παράμετρος της συνάρτησης μας και αυτές είναι η MITER που δίνει στα σχήματα μας οξείς αρθρώσεις, η ROUND που δίνει στα σχήματα μας στρογγυλεμένες αρθρώσεις και τέλος η BEVEL στην οποία οι αρθρώσεις είναι λοξοτομούμενες .Η εξ ορισμού τιμή της συνάρτησης είναι η τιμή MITER.

```
size(480, 120);

smooth();

strokeWeight(12);

strokeJoin(ROUND);

rect(40, 25, 70, 70);

strokeJoin(BEVEL);

rect(140, 25, 70, 70);

strokeJoin(MITER);

rect(240, 25, 70, 70);
```



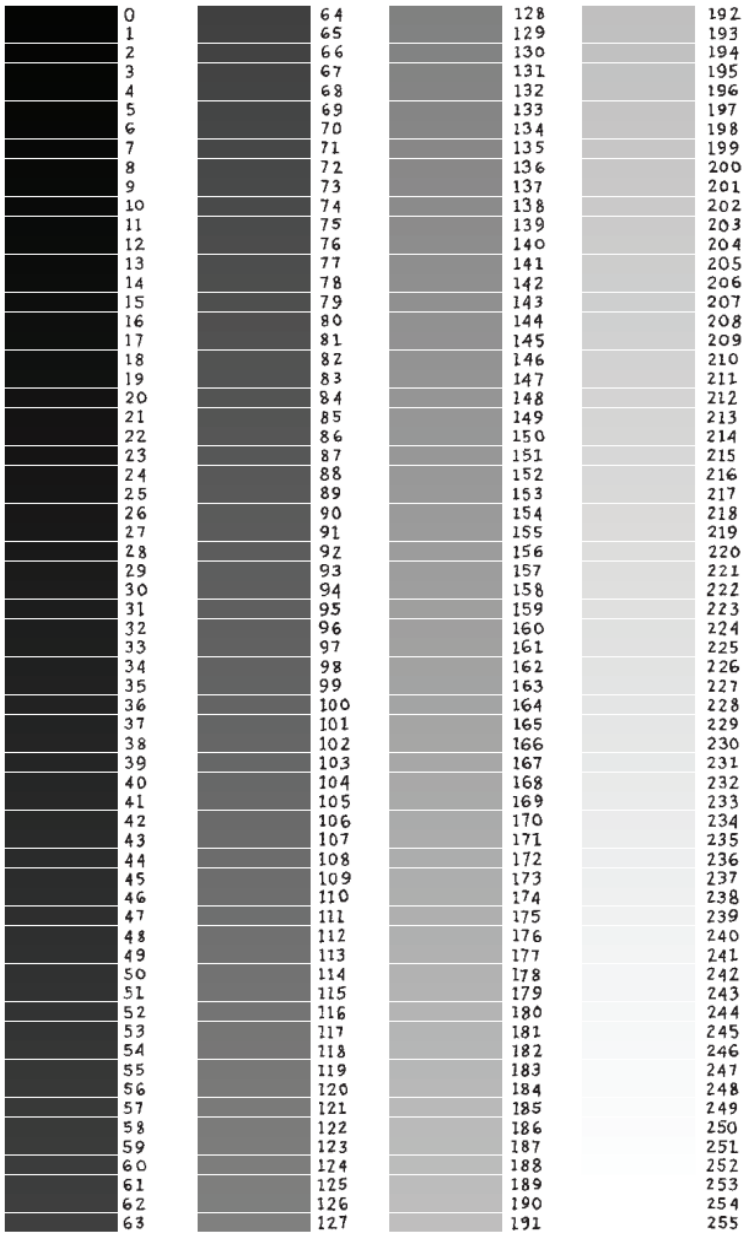
Η συνάρτηση strokeCap() ορίζει το πώς σχεδιάζονται οι άκρες των γραμμών .Υπάρχουν τρεις τρόποι να σχεδιαστούν οι άκρες των γραμμών και αυτοί είναι τετράγωνες, εκτεταμένες και στρογγυλεμένες και ορίζονται αντίστοιχα από τις παρακάτω τιμές: SQUARE, PROJECT, και ROUND.Η εξ ορισμού τιμή της παραμέτρου της συνάρτησης strokeCap() είναι η ROUND.

```
size(480, 120);

strokeWeight(12.0);

strokeCap(ROUND);

line(50, 30, 350, 30);

strokeCap(SQUARE);

line(50, 50, 350, 50);

strokeCap(PROJECT);

line(50, 70, 350, 70);
```



2.6 Το χρώμα των σχημάτων

Όλα τα σχήματα που έχουμε δει μέχρι στιγμής έχουν άσπρο χρώμα μαύρο περίγραμμα και ένα γκρι χρώμα στην επιφάνεια του παραθύρου .Για να τα αλλάξουμε χρησιμοποιούμε τις συναρτήσεις fill(), stroke(),και background() αντίστοιχα .Οι τιμές των παραμέτρων αυτών των συναρτήσεων έχουν εύρος από το 0 έως το 255 όπου το 0 είναι το μαύρο το 255 είναι το άσπρο οι υπόλοιποι αριθμοί είναι αποχρώσεις του γκρι .Στην κλίματα του γκρι έχουμε 256 επιλογές χρωμάτων γιατί αυτή η κλίμακα είναι 8 bits σε αντίθεση με την κλίμακα RGB που είναι μια κλίμακα των 24 bits,θα αναφερθούμε σε αυτήν την κλίμακα παρακάτω. Στην εικόνα που ακολουθεί βλέπουμε αναλυτικά την αντιστοιχία χρώματος και τιμής.



Στην συνέχεια θα ακολουθήσουν μερικά απλά παραδείγματα για να κατανοήσουμε την χρήση των χρωμάτων .Στο πρώτο παράδειγμα θα σχεδιάσουμε ένα απλό σχήμα αλλάζοντας τα χρώματα του και θα τα συγκρίνουμε με τα εξ ορισμού χρώματα της Processing.

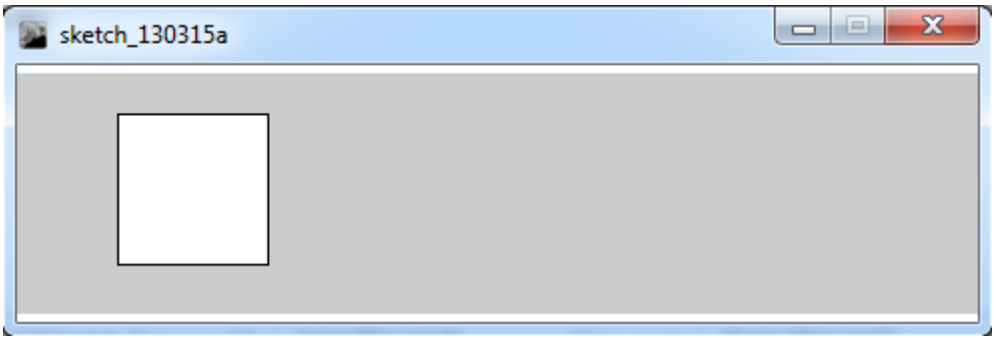
size(480, 120);

background(100);

stroke(255);

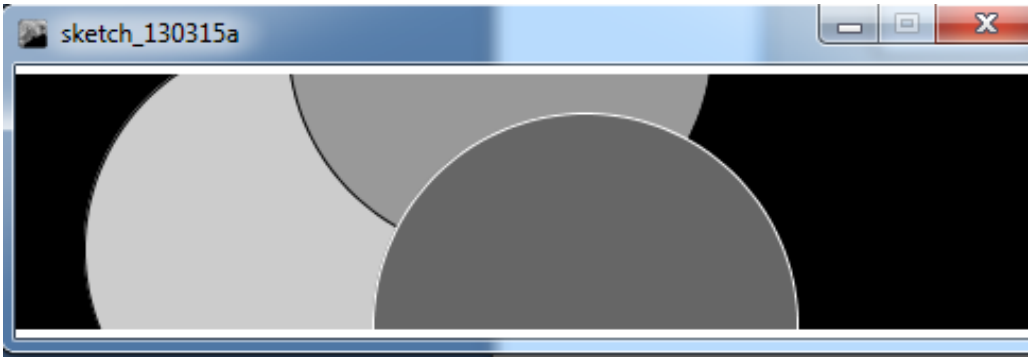
fill(0);

rect(50,20, 75,75);



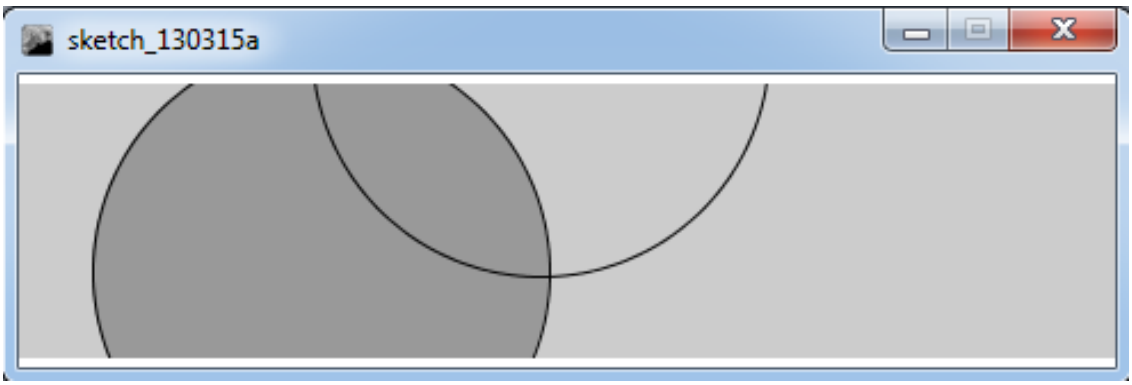
Στο ακόλουθο παράδειγμα θα δούμε τρεις διαφορετικές αποχρώσεις του γκρι στον ίδιο τύπο σχήματος.

```
size(480, 120);

smooth();

background(0);

fill(204);

ellipse(132, 82, 200, 200);

fill(153);

ellipse(228, -16, 200, 200);

fill(102);

stroke(255);

ellipse(268, 118, 200, 200);
```



Στο τρίτο και τελευταίο παράδειγμα θα δούμε πως μπορούμε να απενεργοποιούμε το χρώμα και το περίγραμμα από ένα οποιοδήποτε σχήμα.

```
size(480, 120);

smooth();

fill(153);

ellipse(132, 82, 200, 200);

noFill();

ellipse(228, -16, 200, 200);

noStroke();

ellipse(268, 118, 200, 200); // δεν είναι ορατό
```

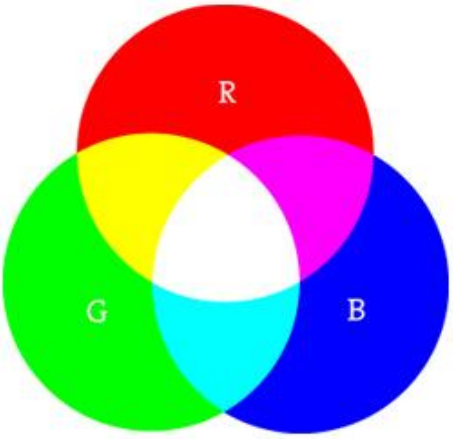


Πρέπει να μην χρησιμοποιούμε τις μεθόδους noStroke() και noFill() ταυτόχρονα γιατί τότε δεν έχουμε οπτικό αποτέλεσμα στην οθόνη μας. Το σχήμα σχεδιάζεται αλλά δεν είναι ορατό στο παραπάνω παράδειγμα .

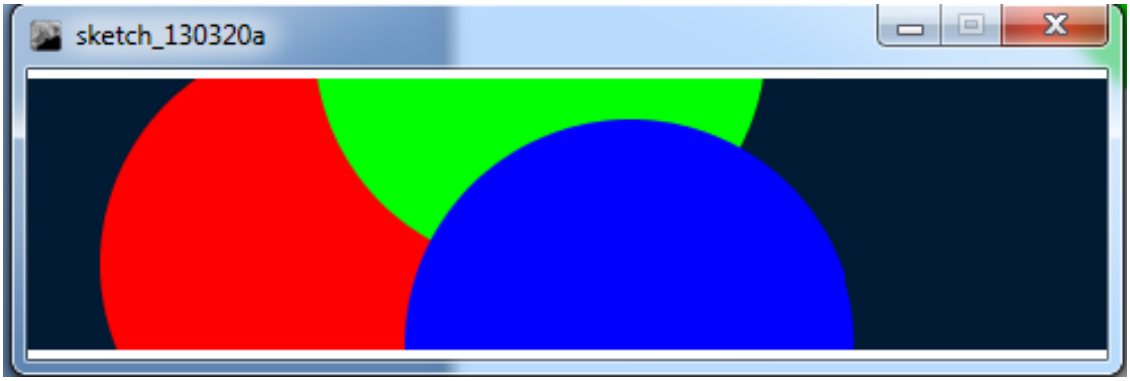
2.6.1 Η κλίμακα χρωμάτων RGB

Μέχρι στιγμής έχουμε ασχοληθεί μόνο με χρώματα που βρίσκονται στην κλίμακα του γκρι. Στην Processing υπάρχει η δυνατότητα να χρησιμοποιήσουμε όλα τα πιθανά χρώματα στα σχήματα μας μέσω της κλίμακας RGB .Η κλίμακα RGB αποτελείται από 3 μεταβλητές των 8bits όπου η καθεμία αντιπροσωπεύει

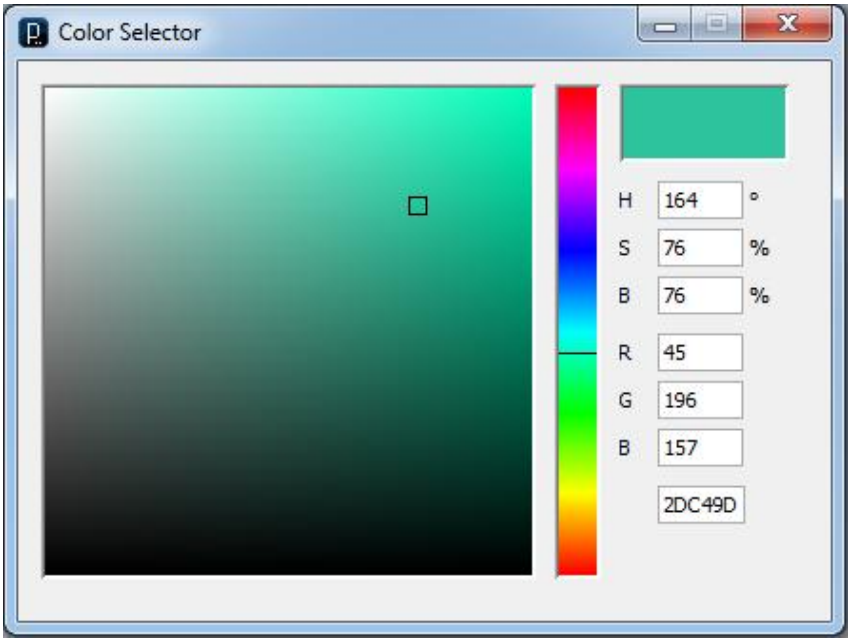
ένα από τα τρία βασικά χρώματα τα οποία είναι το κόκκινο το πράσινο και το μπλε και ο συνδυασμός τους δίνει χρώμα στα σχήματα μας. Θα δούμε ένα πρώτο παράδειγμα για την χρήση των χρωματων.




```
size(480, 120);

noStroke();

smooth();

background(0, 26, 51);

fill(255, 0, 0);

ellipse(132, 82, 200, 200);

fill(0, 255, 0);

ellipse(228, -16, 200, 200);

fill(0, 0, 255);

ellipse(268, 118, 200, 200);
```



Για την εύκολη επιλογή χρωμάτων μπορούμε να χρησιμοποιήσουμε το εργαλείο που υπάρχει στο PDE (Tools→Color Selector) το οποίο είναι μια παλέτα χρωμάτων. Από αυτήν την παλέτα μπορούμε να επιλέξουμε το χρώμα που θέλουμε και να αντιγράψουμε τις τιμές των χρωμάτων RGB στις αντίστοιχες τιμές των συναρτήσεων fill(), stroke(), και background() .



2.6.2 Διαφάνεια χρωμάτων

Για να πετύχουμε διαφάνεια στα χρώματα των σχημάτων μας μπορούμε να χρησιμοποιήσουμε μια τέταρτη παράμετρο που είναι γνωστή σαν alpha. Η παράμετρος αυτή παίρνει τιμές από 0 έως 255, στην τιμή 0 το χρώμα μας γίνεται εντελώς διάφανο , στην τιμή 255 το χρώμα γίνεται εντελώς αδιαφανές και στις τιμές μεταξύ αυτών των δύο άκρων το χρώμα έχει μια κλιμακωτή διαφάνεια.

```
size(200,200);

background(0);

noStroke();

// Δεν έχει οριστεί η μεταβλητή  alpha  άρα το σχήμα είναι αδιαφανές.

fill(0,0,255);

rect(0,0,100,200);

// 100% αδιαφάνεια .

fill(255,0,0,255);

rect(0,0,200,40);

// 75% αδιαφάνεια.

fill(255,0,0,191);

rect(0,50,200,40);

// 55% αδιαφάνεια.
```

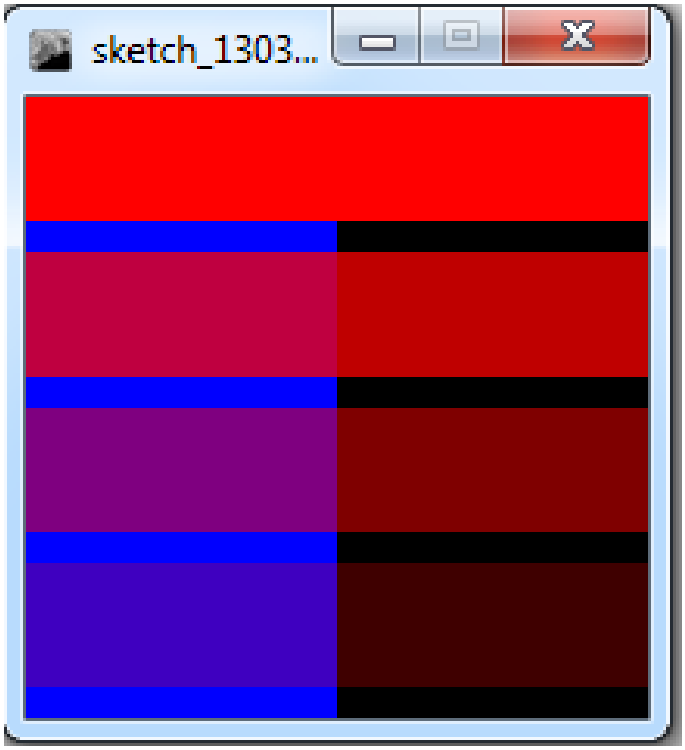
fill(255,0,0,127);

rect(0,100,200,40);

// 25% αδιαφάνεια.

fill(255,0,0,63);

rect(0,150,200,40);



2.7 Σχεδίαση σχημάτων

Δεν είμαστε περιορισμένοι μόνο στα βασικά γεωμετρικά σχήματα μπορούμε να σχεδιάσουμε όποιο σχήμα φανταστούμε ορίζοντας το και συνδέοντας μια σειρά από σημεία. Για να σχεδιάσουμε το τυχαίο σχήμα χρειαζόμαστε τρείς μεθόδους την beginShape() , την vertex() και την endShape(). Με την μέθοδο beginShape() ορίζουμε την αρχή του σχήματος που θέλουμε να σχεδιάσουμε. Με την συνάρτηση vertex() ορίζουμε τις συντεταγμένες των ακραίων σημείων του σχήματος μας. Και τέλος η μέθοδος endShape() ορίζει το τέλος του σχήματος ,αν χρησιμοποιήσουμε την παράμετρο CLOSE στην μέθοδο αυτή ενώνει το τελευταίο ακραίο σημείο με το αρχικό σημείο του σχήματος.

size(480, 120);

beginShape();

vertex(180, 82);

vertex(207, 36);

vertex(214, 63);

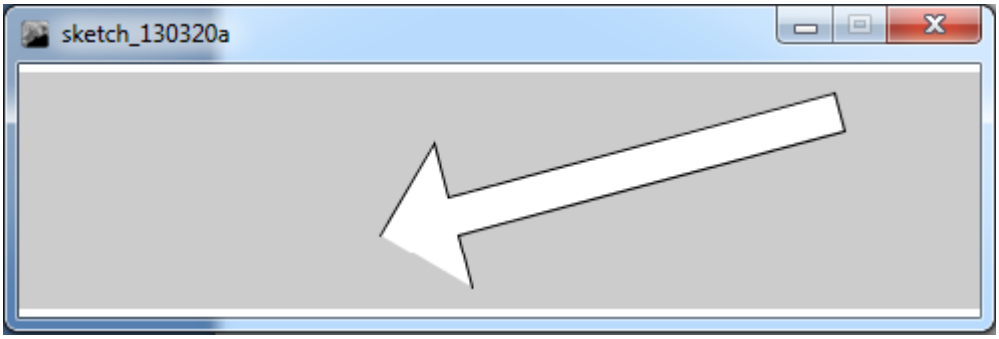
vertex(407, 11);

vertex(412, 30);

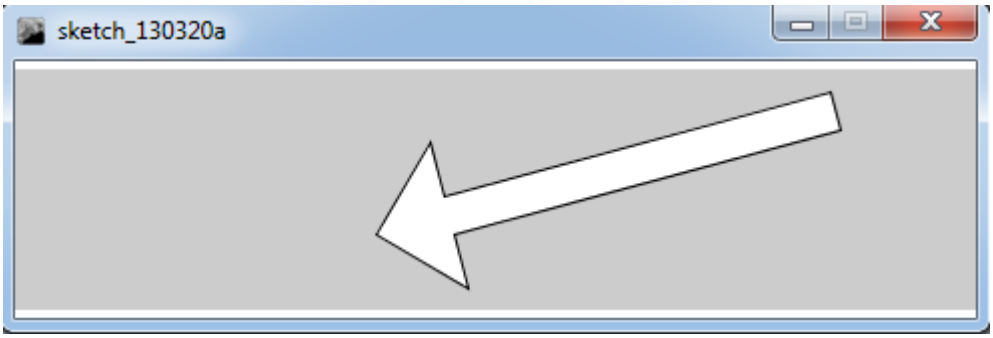
vertex(219, 82);

vertex(226, 109);

endShape();



Αν στο παραπάνω παράδειγμα τροποποιήσουμε την συνάρτηση endShape() και την χρησιμοποιήσουμε με την παράμετρο CLOSE (endShape(CLOSE)) το αποτέλεσμα θα είναι το ακόλουθο



3 Απόκριση

Για να δημιουργήσουμε μια διαδραστική εφαρμογή αυτή θα πρέπει να ανταποκρίνεται στις ενέργειες του χρήστη. Ο κώδικας που είναι υπεύθυνος στο να ανταποκρίνεται στις ενέργειες του χρήστη πρέπει να τοποθετείται στο σώμα της συνάρτησης draw().Οι ενέργειες του χρήστη μπορεί να είναι το πάτημα ενός κουμπιού ή ένα κλικ του ποντικιού . Θα δούμε ένα απλό παράδειγμα της συνάρτησης draw() .

void draw() {

println("I'm drawing");

println(frameCount);

}

Με τον παραπάνω κώδικα παίρνουμε το ακόλουθο αποτέλεσμα.

I'm drawing

1

I'm drawing

2

I'm drawing

3

….

I'm drawing

100

Ο κώδικας που βρίσκεται μέσα στο σώμα της συνάρτησης draw() εκτελείται από την πρώτη μέχρι την τελευταία γραμμή και επαναλαμβάνεται μέχρι να σταματήσουμε το πρόγραμμα η να κλείσουμε το παράθυρο. Κάθε εκτέλεση του κώδικα της συνάρτησης ονομάζεται frame . Ο εξ ορισμού ρυθμός των frames είναι 60 ανά δευτερόλεπτο αλλά υπάρχει δυνατότητα αυτό να αλλάξει με την χρησιμοποίηση της συνάρτησης frameRate(). Επίσης στο παράδειγμα βλέπουμε και την συνάρτηση println() η οποία τυπώνει το κείμενο που της έχουμε ορίσει και των αριθμό των frames.Ο αριθμός των frame υπάρχει αποθηκευμένος στην ειδική μεταβλητή του συστήματος την frameRate. Τα αποτελέσματα της println() τυπώνονται στο κάτω μέρος του PDE οπού εκεί είναι η κονσόλα της Processing.

Σε ένα τυπικό πρόγραμμα της Processing εκτός από την συνάρτηση draw() υπάρχει και η συνάρτηση setup() η οποία εκτελείται μια φορά για να ορίζει κάποιες τιμές μεταβλητών που ισχύουν καθ' όλη την διάρκεια του προγράμματος. Συνήθως μέσα στην συνάρτηση setup() ορίζουμε το μέγεθος του παραθύρου, με την συνάρτηση size() ,το χρώμα του παραθύρου το χρώμα του σχήματος κτλ.

Από τον παραπάνω κώδικα παίρνουμε το εξής αποτέλεσμα

I'm starting

I'm running

...

I'm running

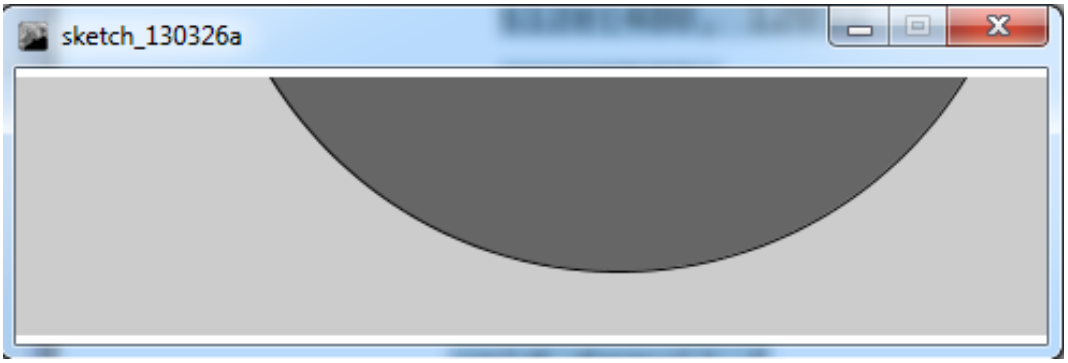
Η εντολή που υπάρχει μέσα στην συνάρτηση draw() συνεχίζει να εκτελείται μέχρι να τερματιστεί το πρόγραμμα.

Μπορούμε να τοποθετήσουμε μεταβλητές εκτός των δύο συναρτήσεων που αναφέρουμε πιο πάνω. Έχουμε αυτήν την δυνατότητα για να μπορούμε να χρησιμοποιούμε τις τιμές των μεταβλητών οπουδήποτε μέσα στο πρόγραμμα. Οι μεταβλητές αυτές ονομάζονται global. Αν ορίσουμε μία μεταβλητή μέσα σε μία συνάρτηση αυτή είναι ορατή μονό στην συνάρτηση και δεν μπορεί να χρησιμοποιηθεί έξω από αυτήν . Αυτός είναι και ο λόγος που χρησιμοποιούμε τις global μεταβλητές. Ταξινομώντας την σειρά που εκτελείται ο κώδικας σε ένα πρόγραμμα της Processing

1. Δημιουργούνται οι global μεταβλητές
2. Εκτελείται η συνάρτηση setup() μια φορά
3. Εκτελείται η συνάρτηση draw() συνεχόμενα μέχρι να τερματιστεί το πρόγραμμα.

Ακολουθεί ένα παράδειγμα για να δούμε τις global μεταβλητές και τις συναρτήσεις setup() και draw() σε ένα πρόγραμμα το οποίο σχεδιάζει μια έλλειψη.

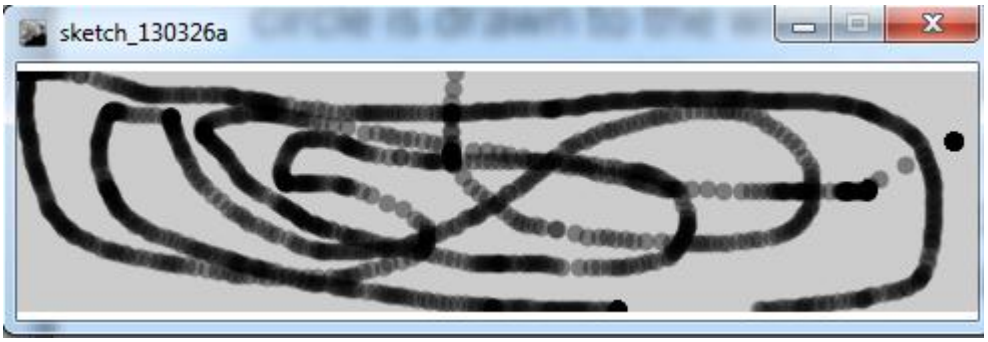
```
int x = 280;

int y = -100;

int diameter = 380;

void setup() {

size(480, 120);

smooth();

fill(102);

}

void draw() {

background(204);

ellipse(x, y, diameter, diameter);

}
```



3.1

Εφόσον έχουμε την δυνατότητα να εκτελούμε κάποια κομμάτια του προγράμματος συνεχόμενα και αφού μπορούμε να εντοπίσουμε και τις συντεταγμένες του ποντικιού μας μπορούμε να τοποθετήσουμε οποιοδήποτε σχήμα στις συγκεκριμένες συντεταγμένες. Μπορούμε να εντοπίζουμε τις συντεταγμένες X,Y του ποντικιού μας από τις μεταβλητές του συστήματος mouseX , mouseY οι οποίες αποθηκεύουν τις συντεταγμένες της θέσης του ποντικιού μας σε κάθε frame.Ας δούμε ένα παράδειγμα όπου θα σχεδιάζουμε έναν κύκλο στην θέση που βρίσκεται το ποντίκι μας.

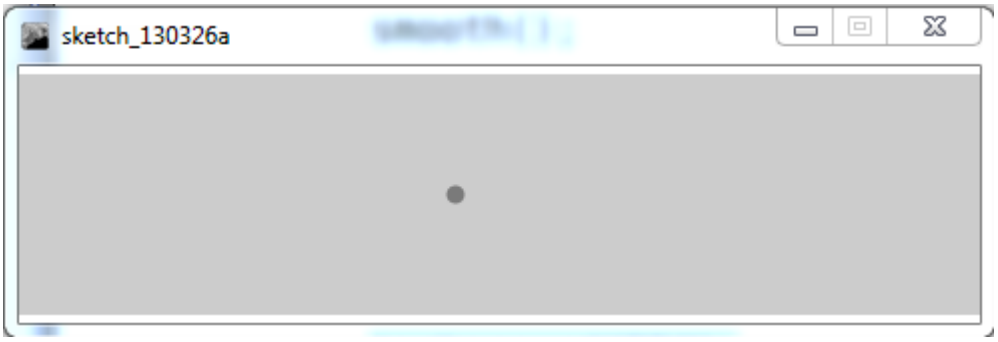
```
void setup() {
size(480, 120);
fill(0, 102);
smooth();
noStroke();
}
void draw() {
ellipse(mouseX, mouseY, 9, 9);
}
```



Η παραπάνω εικόνα σχηματίστηκε κουνώντας το ποντίκι μας στην οθόνη. Όταν οι κύκλοι έχουν απόσταση μεταξύ τους τότε το ποντίκι μας κινείται πιο γρήγορα στην αντίθετη περίπτωση όπου το ποντίκι μας κινείται αργά οι κύκλοι που σχηματίζονται μας δίνουν την εντύπωση της ευθείας γραμμής.

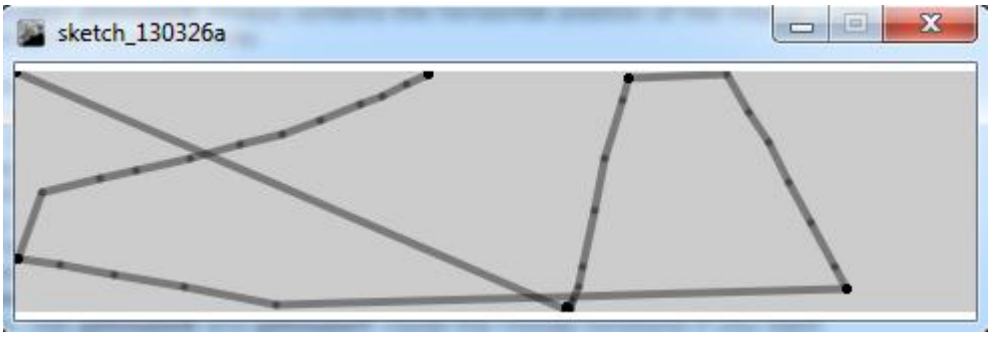
Αν στον παραπάνω κώδικα προσθέσουμε την συνάρτηση background() μέσα στην συνάρτηση draw() θα διαπιστώσουμε ότι η συνάρτηση καθαρίζει το παράθυρο μας. Έτσι έχουμε την αίσθηση ότι ο κύκλος ακολουθεί τον κέρσορα μας .

```
void setup() {
size(480, 120);
fill(0, 102);
smooth();
noStroke();
}
void draw() {
background(204);
ellipse(mouseX, mouseY, 9, 9);
}
```



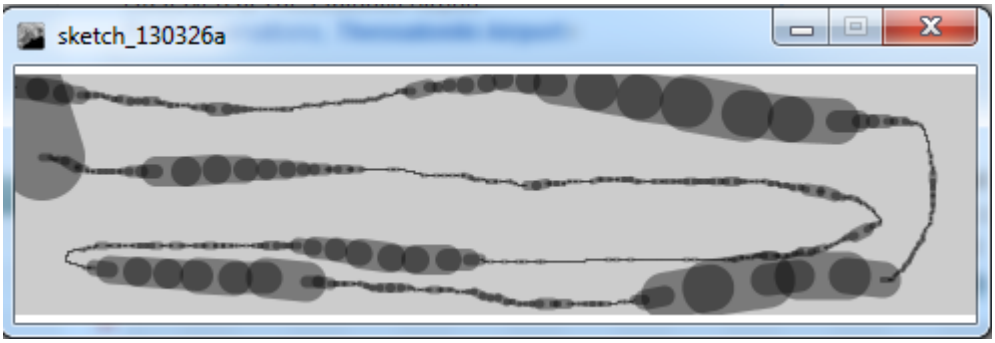
Εκτός από τις μεταβλητές mouseX και mouseY που μας δίνουν τις συντεταγμένες του ποντικιού μας στο τρέχον frame η Processing μας δίνει και τις μεταβλητές pmouseX και pmouseY που μας δίνει τις συντεταγμένες του ποντικιού μας στο προηγούμενο frame.Στο παρακάτω παράδειγμα θα τις συνδυάσουμε για σχεδιάσουμε γραμμές μεταξύ της τρέχουσας και της προηγούμενης θέσης του ποντικιου.

```
void setup() {
size(480, 120);
strokeWeight(4);
smooth();
stroke(0, 102);
}
void draw() {
line(mouseX, mouseY, pmouseX, pmouseY);
}
```



Μπορούμε να χρησιμοποιήσουμε τις μεταβλητές που αναφέραμε για να υπολογίσουμε την ταχύτητα του ποντικιού μας. Αυτό γίνεται υπολογίζοντας την απόσταση της τρέχουσας θέσης του ποντικιού και της προηγούμενης. Αν η απόσταση είναι μικρή τότε το ποντίκι μας κινείται αργά και αν είναι μεγάλη το αντίθετο. Για να υπολογίσουμε αυτήν την απόσταση χρησιμοποιούμε την συνάρτηση dist() η οποία υπολογίζει τη απόσταση μεταξύ δύο σημείων. Θα δώσουμε ένα παράδειγμα όπου η ταχύτητα του ποντικιού μας θα είναι ανάλογη με το πόσο λεπτή θα είναι η γραμμή. Όσο ποιο αργά πηγαίνει το ποντίκι τόσο πιο λεπτή θα είναι η γραμμή και το αντίστροφο.

```
void setup() {
size(480, 120);
smooth();
stroke(0, 102);
}
void draw() {
float weight = dist(mouseX, mouseY, pmouseX, pmouseY);
strokeWeight(weight);
line(mouseX, mouseY, pmouseX, pmouseY);
}
```

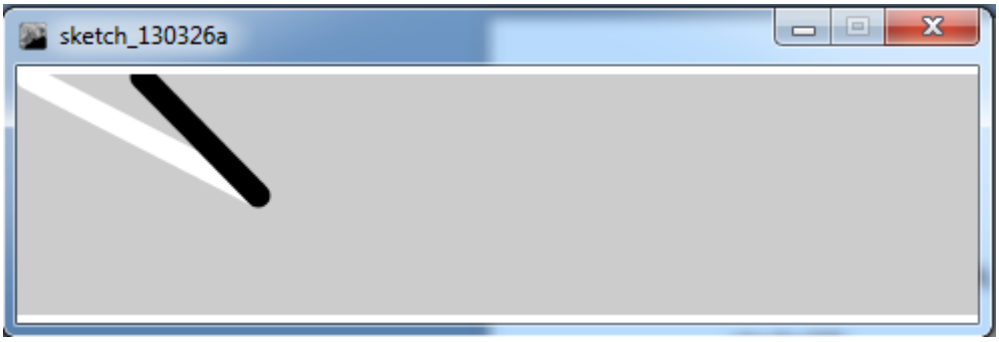


3.2 Καθορισμός Τιμών (mapping)

Όταν χρησιμοποιούμε τιμές για να σχεδιάσουμε ένα οποιοδήποτε σχήμα στην οθόνη μας πολλές φορές είναι χρήσιμο να τις μετατρέπουμε σε κάποιο άλλο εύρος .Για παράδειγμα η μεταβλητή mouseX παίρνει τιμές από 0 μέχρι την τιμή του πλάτους του παραθύρου. Μπορείτε να διαιρέσετε την τιμή του mouseX με έναν αριθμό για να μικρύνετε το εύρος ή μπορείτε να αφαιρέσετε ή να προσθέσετε έναν αριθμό για να μετατοπίσετε το εύρος προς τα αριστερά η προς τα δεξιά αντίστοιχα.

```
void setup() {
size(480, 120);
strokeWeight(12);
smooth();
}
void draw() {
background(204);
```

```
stroke(255);
line(120, 60, mouseX, mouseY); // ασπρή γραμμή
stroke(0);
float temp = mouseX/2 + 60;
line(120, 60, temp, mouseY); // μαύρη γραμμή
}
```

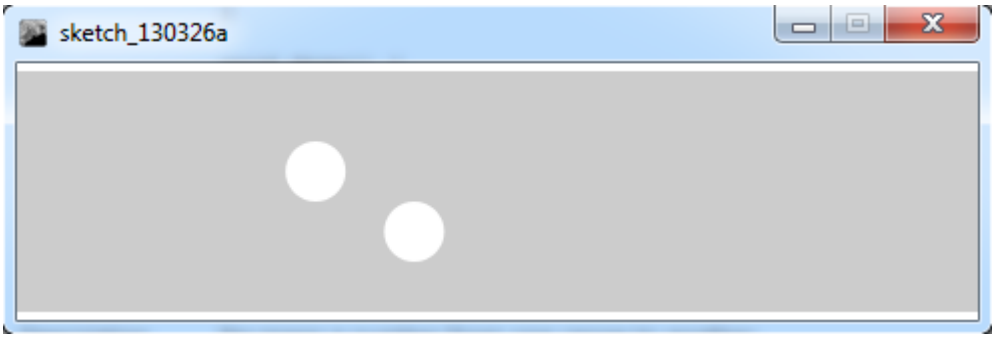


3.2.1 Η συνάρτηση map()

Η Processing μας έχει δώσει την συνάρτηση map() έτσι ώστε να μπορούμε να ορίσουμε εύκολα σε μία μεταβλητή ένα καινούριο εύρος. Η συνάρτηση αυτή έχει 5 παραμέτρους όπου η πρώτη είναι η μεταβλητή της οποίας το εύρος θέλουμε να μετατρέψουμε, η δεύτερη και η τρίτη παράμετρος είναι το κατώτερο και το ανώτερο όριο της μεταβλητής και τέλος η τέταρτη και η πέμπτη παράμετρος είναι το επιθυμητό κατώτερο και ανώτερο όριο. Με την χρήση της συνάρτησης map() γίνεται πιο κατανοητός ο κώδικας μας και το τι θέλουμε να πετύχουμε μέσα από την χρησιμοποίηση γιατί τα όρια του εύρους που θέλουμε να πετύχουμε τα ορίζουμε ως παραμέτρους στην συνάρτηση. Στο παρακάτω παράδειγμα σχεδιάζουμε δύο κύκλους οι οποίοι ακολουθούν το ποντίκι μας στους οποίους έχουμε ορίσει ένα καινούριο εύρος για τον καθένα όσον αναφορά τις τιμές της μεταβλητής mouseX.

```
void setup() {
  size(480, 120);
  noStroke();
}
void draw() {
  background(204);
  float  x1 = map(mouseX, 0, width, 50, 150);
  ellipse(x1, 50, 30, 30);
  float x2 = map(mouseX, 0, width, 0, 200);
```

```
  ellipse(x2, 80, 30, 30);
}
```

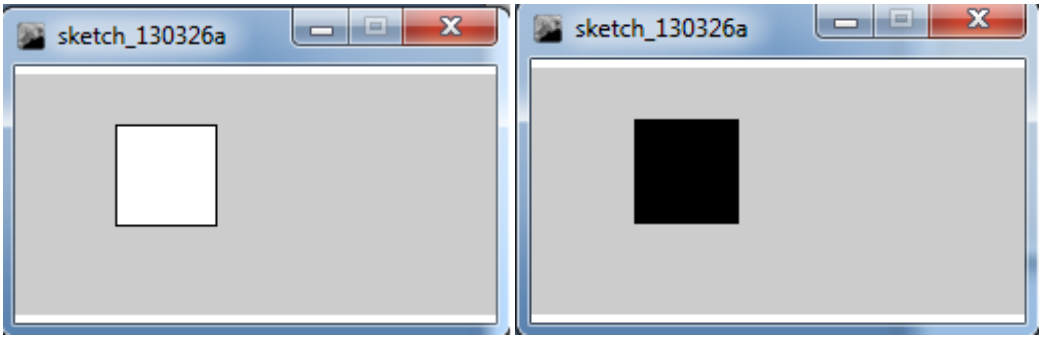


3.3 κλικ

3.3.1 Η μεταβλητή mousePressed

Η Processing εκτός από το να αποθηκεύει τις συντεταγμένες της θέσης του ποντικιού στις μεταβλητές του συστήματος mouseX και mouseY διαθέτει και μια επιπλέον μεταβλητή συστήματος την mousePressed η οποία εντοπίζει αν κάποιο κουμπί του ποντικιού έχει πατηθεί .Η μεταβλητή αυτή είναι τύπου Boolean αρά μπορεί να έχει δύο τιμές ανάλογα αν κάποιο κουμπί πατηθεί η όχι. Θα δούμε ένα απλό παράδειγμα όπου το σχήμα θα αλλάζει χρώμα ανάλογα αν είναι πατημένο το κουμπί του ποντικιού μας.

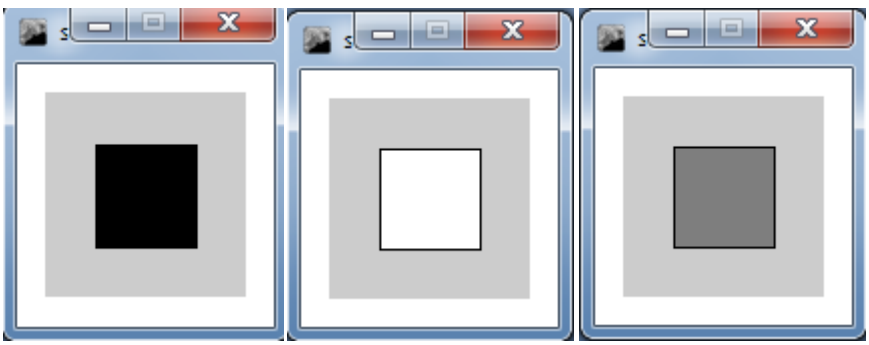
```
void setup() {
size(240, 120);
}
void draw() {
  if (mousePressed == true) {
    fill(0);
  } else {
    fill(255);
  }
  rect(50, 25, 50, 50);
}
```



3.3.2 Η μεταβλητή mouseButton

Μόλις πατηθεί ένα κουμπί του ποντικιού μας η Processing αποθηκεύει στη μεταβλητή mouseButton μια από τις τιμές LEFT, RIGHT, ή CENTER ανάλογα με το ποιο κουμπί πατήθηκε. Στο ακόλουθο παράδειγμα βλέπουμε ένα τετράγωνο το οποίο παίρνει μαύρο χρώμα αν πατηθεί το αριστερό κουμπί ,λευκό χρώμα αν πατηθεί το δεξί κουμπί και στις υπόλοιπες περιπτώσεις το σχήμα μας έχει ένα γκρι χρώμα.

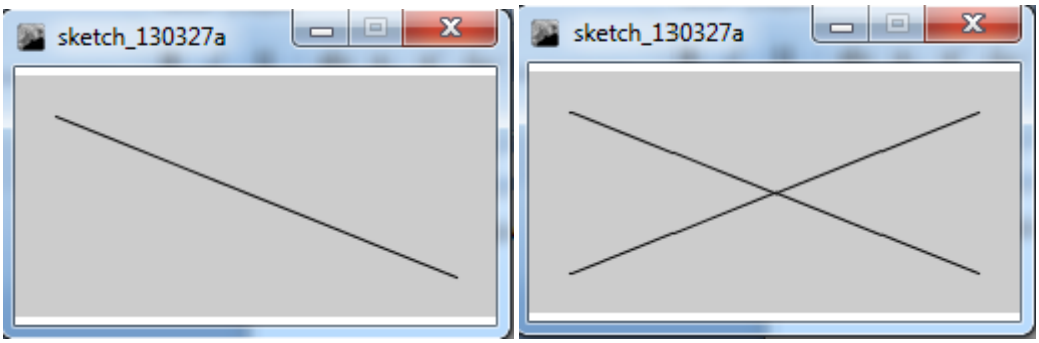
```
void draw() {
 if ((mousePressed == true ) && (mouseButton == LEFT)) {
   fill(0);
 } else if ( (mousePressed == true ) && (mouseButton == RIGHT)) {
   fill(255);
 } else {
   fill(126);
 }
 rect(25, 25, 50, 50);
}
```



3.4 Διάδραση με το πληκτρολόγιο

Η Processing έχει την δυνατότητα να εντοπίζει αν κάποιο πλήκτρο του πληκτρολογίου μας είναι πατημένο , χρησιμοποιώντας την μεταβλητή keyPressed η οποία είναι ίδιου τύπου με την μεταβλητή mousePressed . Η μεταβλητή keyPressed είναι τύπου Boolean και παίρνει την τιμή true όσο είναι κάποιο κουμπί είναι πατημένο, ενώ παίρνει την τιμή false στην αντίθετη περίπτωση. Θα δούμε ένα παράδειγμα όπου σε ένα παράθυρο είναι σχεδιασμένη μία γραμμή, και μια δεύτερη γραμμή θα σχεδιάζεται στο παράθυρό μας, όταν πατήσουμε κάποιο κουμπί του πληκτρολογίου .

```
void setup() {

size(240, 120);

smooth();

}

void draw() {

background(204);

line(20, 20, 220, 100);

if (keyPressed) {

line(220, 20, 20, 100);

}
```

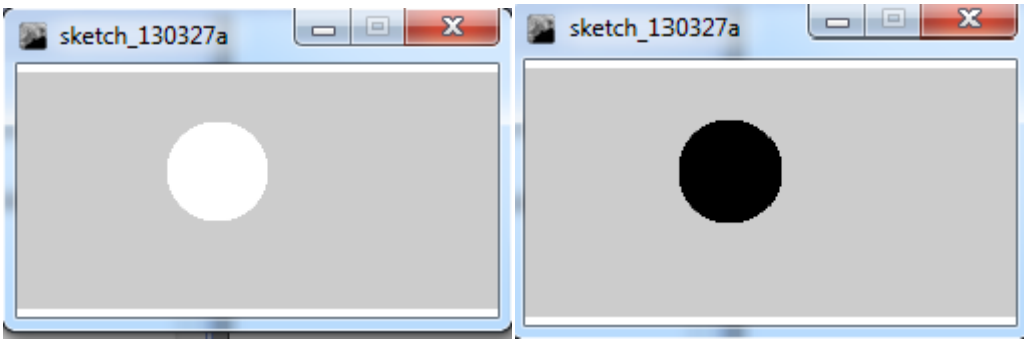


3.4.1 Η μεταβλητή key

Η μεταβλητή key αποθηκεύει το πιο πρόσφατο κουμπί του πληκτρολογίου που έχει πατηθεί .Η μεταβλητή key είναι τύπου char οπότε μπορεί να αποθηκεύσει απλούς χαρακτήρες ,όπως γράμματα, αριθμούς και σύμβολα. Σε αντίθεση με την μεταβλητή keyPressed η οποία χάνει την τιμή της μόλις το κουμπί ελευθερωθεί, η key κρατάει την τιμή της μέχρι να πατηθεί κάποιο άλλο κουμπί. Στο παράδειγμα που ακολουθεί βλέπουμε ότι το σχήμα χρωματίζεται αν πατηθεί ένα από τα κουμπιά b ή B και μένει χρωματισμένο μέχρι να πατηθεί κάποιο άλλο κουμπί.

```
void setup() {
```

```
size(240, 120);
smooth();
}
void draw() {
   if (key == 'b' || key == 'B') {
     fill(0);
   }
  else {
   fill(255);
  }
  ellipse(100, 25, 50, 50);
}
```



Μερικά πλήκτρα του πληκτρολογίου , είναι πιο δύσκολα να εντοπίσουμε γιατί δεν αντιστοιχούν σε κάποιο συγκεκριμένο γράμμα ή αριθμό . Αυτά τα πλήκτρα είναι τα βέλη του πληκτρολογίου ,το shift, το alt, το ctrl για τα οποία χρειαζόμαστε ένα επιπλέον βήμα, για να καταλάβουμε αν έχουν πατηθεί η όχι. Η Processing ομαδοποιεί όλα αυτά τα πλήκτρα σε μια κατηγορία και τα ονομάζει coded. Πρώτα πρέπει να ελέγξουμε αν το κουμπί που πατήθηκε ανήκει σε αυτήν την κατηγορία. Έπειτα με την βοήθεια της μεταβλητής keyCode πρέπει να ελέγξουμε πιο από αυτά τα κουμπιά πατήθηκε. Οι τιμές που συναντάμε συχνότερα για την μεταβλητή keyCode είναι η ALT,η CONTROL, και η SHIFT, επίσης είναι οι τιμές που αντιπροσωπεύουν τα βέλη του πληκτρολογίου, UP, DOWN, LEFT, and RIGHT.Στο ακόλουθο παράδειγμα μετακινούμε το σχήμα μας δεξιά ή αριστερά ανάλογα ποιο κουμπί του πληκτρολογίου πατάμε.

```
int x = 215;
void setup() {
size(480, 120);
}
void draw() {
```

```
background(190);
if (keyPressed && (key == CODED)) {
if (keyCode == LEFT) {
x--;
} else if (keyCode == RIGHT) {
x++;
}
}
rect(x, 45, 50, 50);
}
```

4

Η Processing μας δίνει την δυνατότητα να κάνουμε περισσότερα πράγματα από το να σχεδιάζουμε γραμμές και απλά σχήματα. Μπορούμε να προσθέσουμε ψηφιογραφικές εικόνες, διανυσματικές εικόνες ,γραμματοσειρές και διανυσματικά σχήματα ώστε να επεκτείνουμε τις οπτικές δυνατότητες των προγραμμάτων. Η Processing χρησιμοποιεί έναν φάκελο, που τον ονομάζει data για να αποθηκεύει τα συγκεκριμένα αρχεία. Αυτό έχει ως αποτέλεσμα να μην χρειάζεται να ανησυχούμε για τις διαδρομές των αρχείων που ενσωματώνουμε στο πρόγραμμα μας.

4.1 Εικόνες

Για να προσθέσουμε μια εικόνα στο πρόγραμμα μας πρέπει να ακολουθήσουμε τα παρακάτω τρία βήματα:

1. Εισαγωγή της εικόνας στον φάκελο data.

2.Δημιουργία ενός αντικειμένου PImage.

3.Φόρτωση στης εικόνας στο αντικείμενο PImage με την χρήση της μεθόδου loadImage().

Μπορούμε να εισάγουμε την εικόνα στο φάκελο data με δύο τρόπους. Είτε σέρνοντας την εικόνα στη επιφάνεια του PDE, είτε χρησιμοποιώντας το menu του PDE την εντολή Sketch→ Add File. Σε περίπτωση που ο φάκελος data δεν υπάρχει, δημιουργείται αυτόματα μόλις αντιγράψουμε ένα αρχείο.

Για να εμφανίσουμε την εικόνα στη οθόνη εκτός από τα παραπάνω τρία βήματα πρέπει να χρησιμοποιήσουμε την μέθοδο image().Η μέθοδος image έχει πέντε παραμέτρους, η πρώτη παράμετρος καθορίζει ποια εικόνα θα σχεδιαστεί στην οθόνη ,η δεύτερη και η τρίτη παράμετρος είναι οι συντεταγμένες X,Y του σημείου που εμφανίζεται η εικόνα μας και τέλος η τέταρτη και η πέμπτη παράμετρος είναι το πλάτος και το ύψος της εικόνας που εμφανίζεται. Αν δεν οριστούν οι δύο τελευταίες παράμετροι στην οθόνη

μας εμφανίζεται το πραγματικό μέγεθος της εικόνας. Στην συνέχεια θα δούμε τρία παραδείγματα εμφάνισης εικόνων .Στο πρώτο εμφανίζουμε απλά μια εικόνα στην οθόνη μας , στο δεύτερο παράδειγμα εμφανίζουμε δύο εικόνες και τέλος στο τρίτο παράδειγμα εμφανίσουμε μια εικόνα στην οθόνη μας και το μέγεθος της μεταβάλλεται όσο το ποντίκι μας κινείται .

1ο Παράδειγμα

```
PImage img;

void setup() {

size(480, 120);

img = loadImage("land_1.jpg");

}

void draw() {

image(img,0,0);

}
```

2ο Παράδειγμα

```
PImage img1;

PImage img2;

void setup() {

size(480, 120);

img1 = loadImage("land_1.jpg");

img2 = loadImage("land_2.jpg");

}

void draw() {

image(img1,-300,0);

image(img2, 180, 0, 240, 120);

}
```

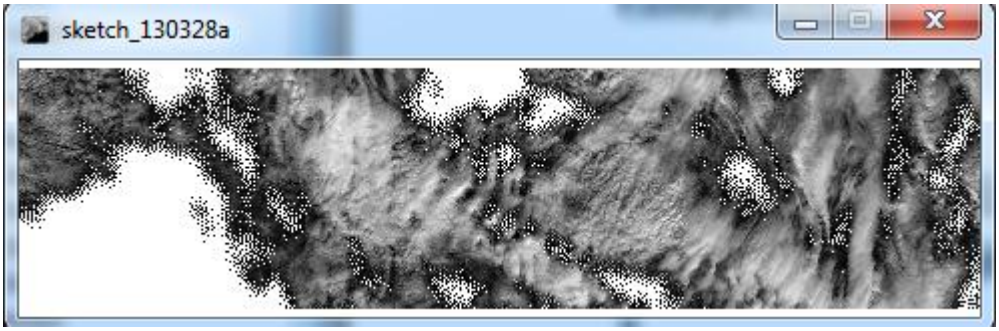
3ο Παράδειγμα

```
PImage img2;

void setup() {

size(480, 120);

img2 = loadImage("land_2.jpg");

}

void draw() {

background(0);

image(img2, 0, 0, mouseX, mouseY);

}
```

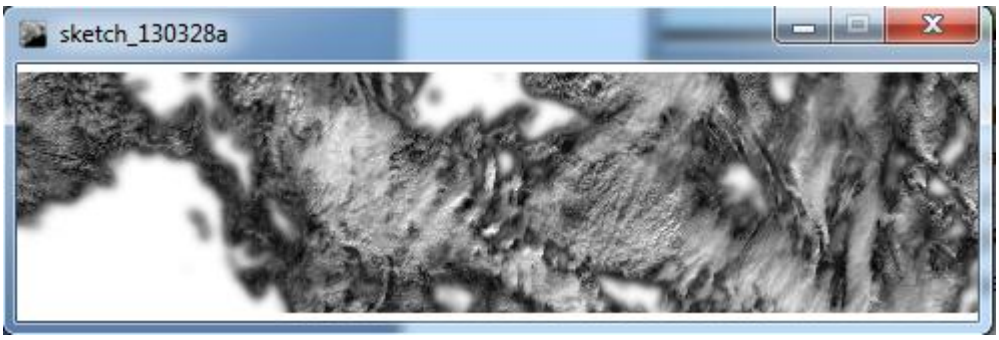
Οι εικόνες τύπου PNG και GIF υποστηρίζουν την διαφάνεια. Οι εικόνες τύπου GIF διαθέτουν ένα pixel για την διαφάνεια ,πράγμα που σημαίνει ότι αυτό το pixel είναι είτε διάφανο είτε αδιάφανο. Οι εικόνες τύπου PNG διαθέτουν 8 pixels για την διαφάνεια το οποίο σημαίνει ότι κάθε ένα από αυτά τα 8 pixels μπορούν να έχουν ένα μεταβλητό επίπεδο διαφάνειας. Θα δούμε δύο παραδείγματα ένα για κάθε τύπο εικόνας για να κατανοήσουμε την διαφορά.

```
PImage img;

void setup() {

size(480, 120);

img = loadImage("clouds.gif");

}
```

```
void draw() {
background(255);
image(img, 0, 0);
image(img, 0, mouseY * -1);
}
```

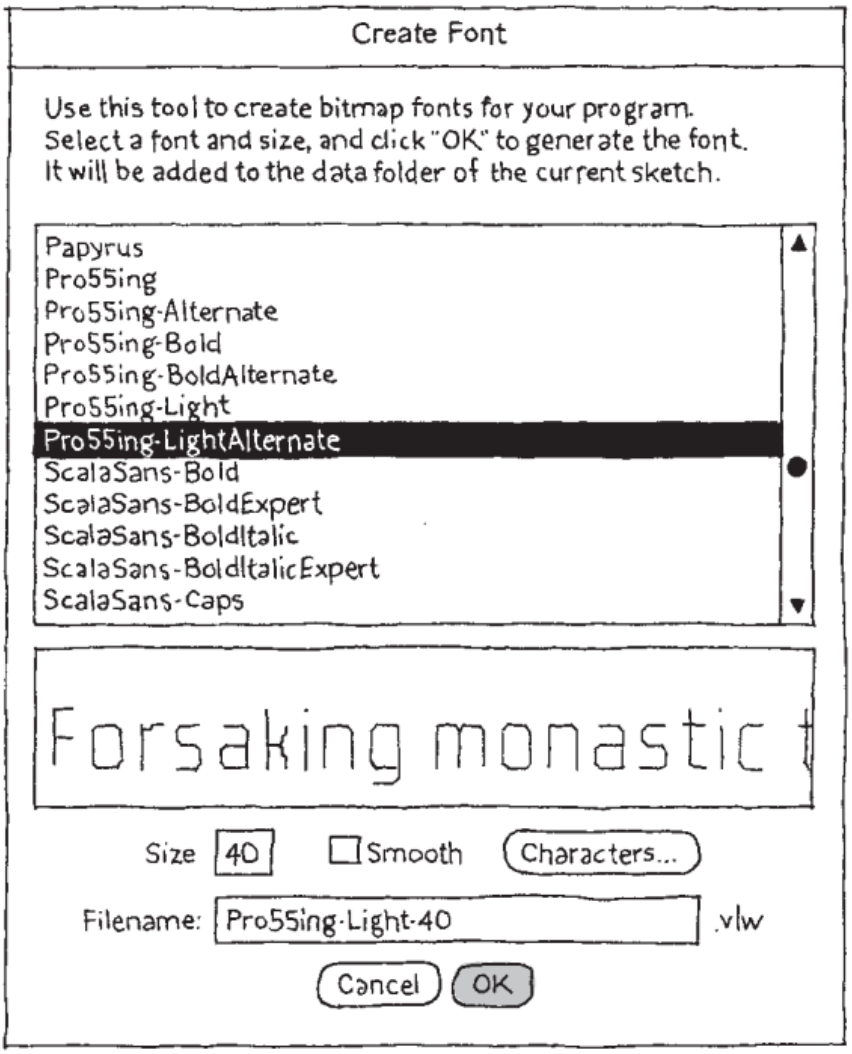



```
PImage img;
void setup() {
size(480, 120);
img = loadImage("clouds.png");
}
void draw() {
background(255);
image(img, 0, 0);
image(img, 0, mouseY * -1);
}
```



4.2 Γραμματοσειρές

Η Processing μπορεί να εμφανίσει κάποιο κείμενο σε πολλές γραμματοσειρές, εκτός από την γραμματοσειρά που έχει η γλώσσα εξ ορισμού. Για να χρησιμοποιήσουμε τις γραμματοσειρές που διαθέτει ο υπολογιστής μας, πρέπει πρώτα να δημιουργήσουμε ένα αρχείο τύπου VLW. Το αρχείο αυτού του τύπου αποθηκεύει κάθε χαρακτήρα της γραμματοσειράς σαν μία μικρή εικόνα. Για να μετατρέψουμε την γραμματοσειρά χρησιμοποιούμε το εργαλείο του PDE που το βρίσκουμε στο menu Tools→Create Font. Επιλέγουμε την γραμματοσειρά που θέλουμε και δημιουργούμε το αρχείο με κατάληξη VLW το οποίο αποθηκεύεται στον φάκελο data.



Για να προσθέσουμε την νέα γραμματοσειρά, πρέπει να ακολουθήσουμε μια παρόμοια διαδικασία με αυτήν που ακολουθούμε στην εισαγωγή εικόνων. Πρέπει να ακολουθήσουμε τα εξής βήματα:

1. Δημιουργούμε το αρχείο VLW .

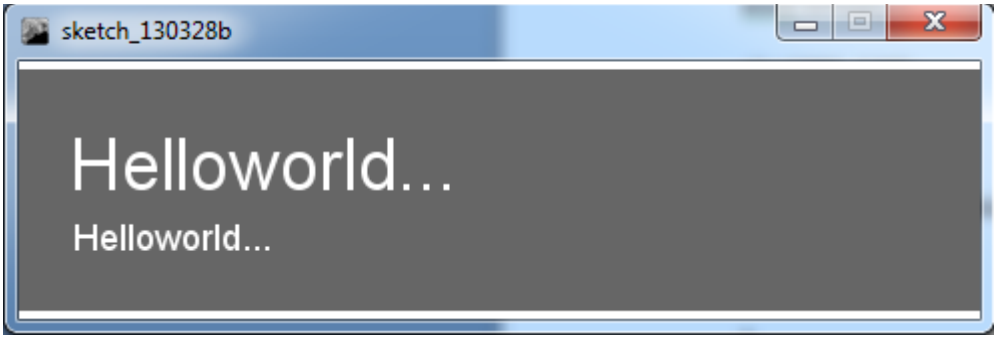
2. Δημιουργούμε ένα αντικείμενο τύπου PFont.

3. Φορτώνουμε την γραμματοσειρά με την μέθοδο loadFont().

4. Χρησιμοποιούμε την μέθοδο textFont() για να ορίσουμε την συγκεκριμένη γραμματοσειρά.

Για να γράψουμε ένα κείμενο στην οθόνη μας χρησιμοποιούμε την συνάρτηση text() και για να ορίσουμε το μέγεθος της γραμματοσειράς την συνάρτηση textSize(). Η πρώτη παράμετρος της μεθόδου text() είναι οι χαρακτήρες που θέλουμε να γράψουμε μέσα σε διπλά εισαγωγικά ,η τρίτη και η τέταρτη παράμετρος ,είναι οι συντεταγμένες X,Y της τοποθεσίας του αλφαριθμητικού στην οθόνη μας. Η τοποθεσία είναι σχετική με την γραμμή της βάσης του κειμένου. Στο παρακάτω παράδειγμα θα φορτώσουμε μία γραμματοσειρά στον κώδικα μας και θα γράψουμε ένα απλό κείμενο.

```
PFont font;

void setup() {

size(480, 120);

smooth();

font = loadFont("ArialMT-48.vlw");

textFont(font);

}

void draw() {

background(102);

textSize(36);

text("Helloworld...", 25, 60);

textSize(18);

text("Helloworld...", 27, 90);

}
```



4.3 Σχήματα

Αν έχετε κατασκευάσει μια διανυσματικό σχήμα ή μια διανυσματική εικόνα σε κάποιο πρόγραμμα όπως το adobe lllustrator, μπορείτε να το φορτώσετε απευθείας στην Processing .Αυτό είναι χρήσιμο για

σχήματα που δεν είναι δυνατό να σχεδιαστούν με τις μεθόδους της Processing . Όπως και οι εικόνες έτσι και τα διανυσματικά σχήματα πρέπει να προστεθούν στον φάκελο data ώστε να μπορούν να φορτωθούν στο πρόγραμμα μας. Έπειτα δημιουργούμε ένα αντικείμενο τύπου PShape , ώστε να μπορούμε να αποθηκεύσουμε το διανυσματικό σχήμα. Τέλος με την μέθοδο loadShape() φορτώνουμε το διανυσματικό σχήμα στον κώδικα μας. Για να εμφανίσουμε το διανυσματικό σχήμα στην οθόνη μας χρησιμοποιούμε την μέθοδο shape() η οποία έχει την ίδια λειτουργία με την μέθοδο image(). Παρακάτω θα δούμε ένα παράδειγμα φόρτωσης και εμφάνισης ενός διανυσματικού σχήματος.

```
PShape network;

void setup() {

size(480, 120);

smooth();

network = loadShape("network.svg");

}

void draw() {

background(0);

shape(network, 30, 10);

shape(network, 180, 10, 280, 280);

}
```

5 Κίνηση

Όταν μια ομάδα παρόμοιων εικόνων εναλλάσσεται με ένα αρκετά γρήγορο ρυθμό στο μυαλό μας δημιουργείται η αίσθηση της κίνησης. Η Processing για να δημιουργήσει την αίσθηση της κίνησης, εκτελεί τον κώδικα που βρίσκεται μέσα στην συνάρτηση draw() 60 φορές το δευτερόλεπτο. Στην πραγματικότητα η Processing προσπαθεί να τρέξει τον κώδικα που βρίσκεται μέσα στην draw() 60 φορές σε ένα δευτερόλεπτο , αν δεν τα καταφέρει μειώνει τον ρυθμό των

frames ανά δευτερόλεπτο. Η συνάρτηση frameRate() ορίζει τον μέγιστο ρυθμό frames ανά δευτερόλεπτο, ο πραγματικός ρυθμός των frames εξαρτάται από την υπολογιστική ισχύς του υπολογιστή που εκτελεί το πρόγραμμα. Η μεταβλητή frameRate παρακολουθεί την ταχύτητα του προγράμματος μας. Στο παρακάτω παράδειγμα ορίζουμε έναν άλλο ρυθμό frames και τυπώνουμε στην κονσόλα του PDE την ταχύτητα του προγράμματος μας.

```
void setup() {
  size(480,120);
  frameRate(30);
}
void draw() {
  line(0, 0, width, height);
  println(frameRate);
}
```

5.1 Διεύθυνση και Ταχύτητα

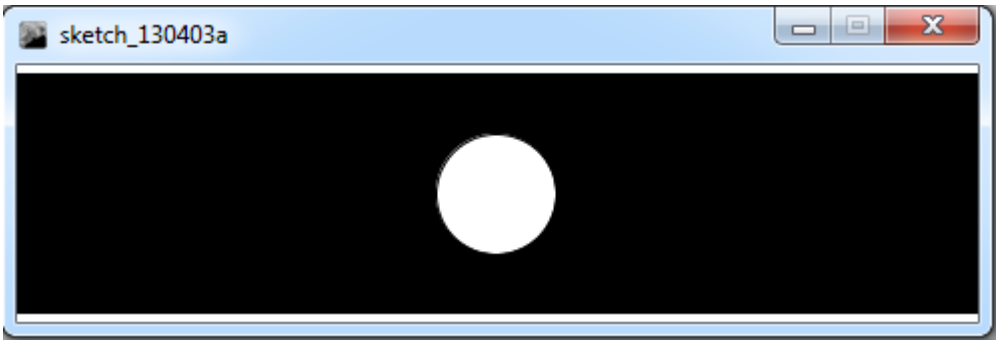
Για να δημιουργήσουμε μια αίσθηση μιας πιο φυσικής κίνησης θα προτιμήσουμε τις μεταβλητές τύπου float.Προτιμούμε τις float μεταβλητές για την δυνατότητα που μας δίνουν να αποθηκεύουμε δεκαδικές τιμές, προσφέροντας μας μεγαλύτερη ανάλυση που μας βοηθά να δημιουργούμε κίνηση. Παρακάτω θα δώσουμε μερικά παραδείγματα κίνησης και διεύθυνσης ενός σχήματος.

1ο Παράδειγμα

Σε αυτό το παράδειγμα θα δούμε την οριζόντια κίνηση μιας σφαίρας. Η ταχύτητα της σφαίρας ορίζεται από την μεταβλητή speed. Αν αυξήσουμε την τιμή αυτής της μεταβλητής θα αυξηθεί και η ταχύτητα της σφαίρας.

```
int radius = 40;
float x = -radius;
float speed = 0.5;
void setup() {
size(480, 120);
smooth();
ellipseMode(RADIUS);
```

```
}
void draw() {
background(0);
x += speed;
ellipse(x,60,radius,radius);
}
```



2ο Παράδειγμα

Σε αυτό το παράδειγμα θα τροποποιήσουμε τον παραπάνω κώδικα έτσι ώστε η σφαίρα μας να επιστρέφει στην αρχική της θέση , μόλις η σφαίρα εξαφανιστεί από την οθόνη μας. Αυτό θα το πετύχουμε προσθέτοντας μια συνθήκη , όπου ελέγχουμε αν το κέντρο του κύκλου έχει απομακρυνθεί σε απόσταση μεγαλύτερη από το πλάτος του παραθύρου μας συν την ακτίνα της σφαίρας.

```
int radius = 40;
float x = -radius;
float speed = 0.5;
void setup() {
size(480, 120);
```

```
smooth();
ellipseMode(RADIUS);
}
void draw() {
background(0);
x += speed;
if (x > width+radius) {
x = -radius;
}
ellipse(x,60,radius,radius);
}
```

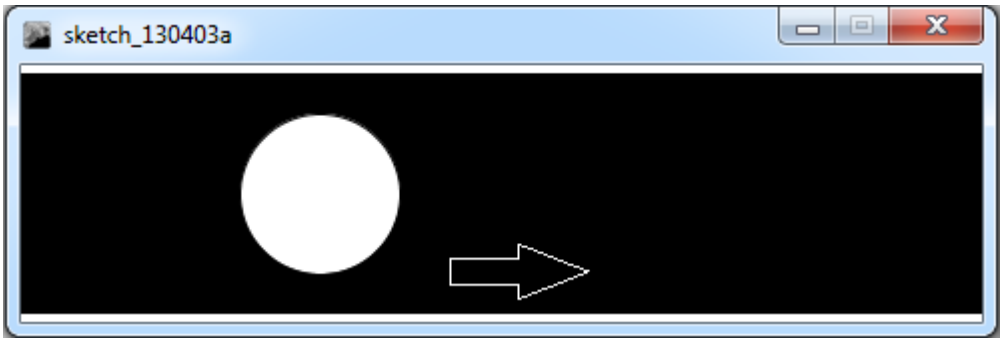
Το οπτικό αποτέλεσμα είναι όμοιο με το πρώτο παράδειγμα.

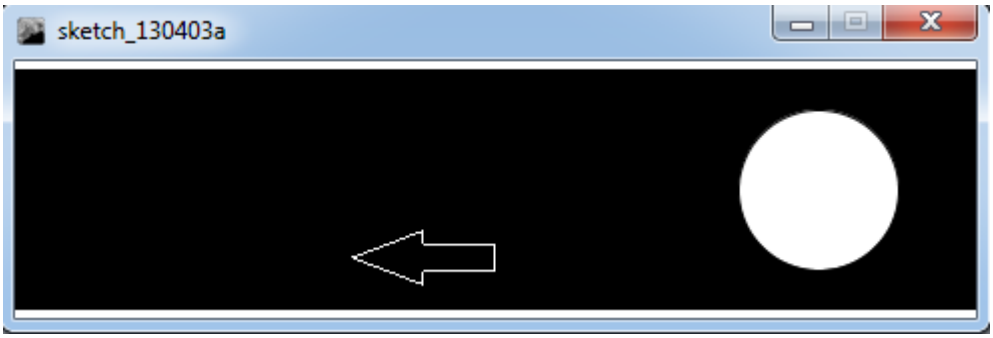
3ο Παράδειγμα

Σε αυτό το παράδειγμα η σφαίρα μας αλλάζει κατεύθυνση όταν εφάπτεται στα άκρα του παραθύρου. Αυτό το πετυχαίνουμε την χρήση της μεταβλητής direction η οποία ορίζει την διεύθυνση του σχήματος

```
int radius = 40;
float x = 40;
float speed = 0.5;
int direction = 1;
void setup() {
size(480, 120);
smooth();
ellipseMode(RADIUS);
}
void draw() {
background(0);
```

```
x += speed * direction;
if ((x > width-radius) || (x < radius)) {
direction = -direction;
}
ellipse(x,60,radius,radius);
}
```





Τα βέλη μας δείχνουν την διεύθυνση της σφαίρας δεν αποτελούν μέρος του προγράμματος.

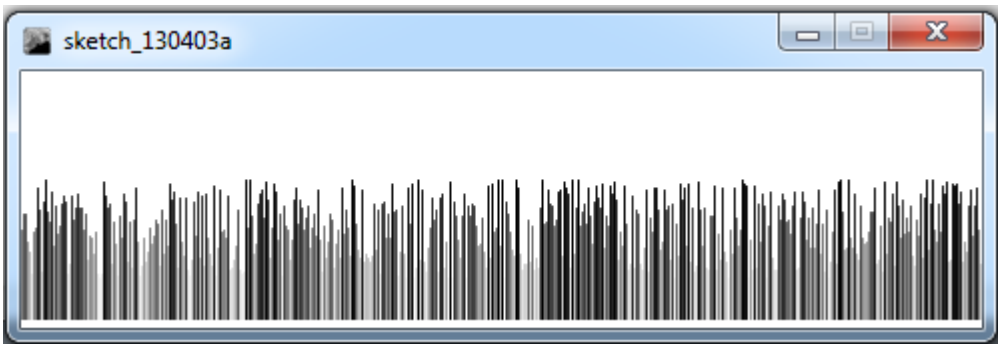
5.2 Τυχαίες τιμές

Οι περισσότερες κινήσεις των γραφικών των υπολογιστών είναι ευθείες και ομαλές κινήσεις. Στην περίπτωση που θέλουμε να προσομοιώσουμε μια τυχαία κίνηση στον χώρο, θα χρειαστούμε τυχαίες τιμές. Αυτές τις τιμές μπορούμε να τις πάρουμε με την βοήθεια της συνάρτησης random(). Η συνάρτηση random() μπορεί να συνταχτεί είτε με μία είτε με δύο παραμέτρους. Όταν συντάσσεται με μια παράμετρο , η συνάρτηση μας επιστρέφει τυχαίες τιμές μεταξύ του 0 και της τιμής που ορίσαμε. Αν την συντάξουμε με δύο παραμέτρους, τότε η συνάρτηση random() μας επιστρέφει τυχαίες τιμές έχοντας ως κατώτατο όριο την πρώτη παράμετρο και ως ανώτατο όριο την δεύτερη παράμετρο. Οι τιμές που επιστρέφει η συνάρτηση μας είναι τύπου float.Παρακάτω θα δούμε δύο παραδείγματα όπου θα χρησιμοποιήσουμε την συνάρτηση random().

1ο Παράδειγμα

Σε αυτό το παράδειγμα σχεδιάζουμε τυχαίου μήκους και χρώματος κάθετες γραμμές.

```
 void setup(){
 size(480,120);
 smooth();
 frameRate(1);
 }
void draw(){
 background(255);
for (int i = 0; i < width; i++) {
 float r = random(50);
 stroke(r*5);
 line(i, height, i, 50+r);
}
}
```

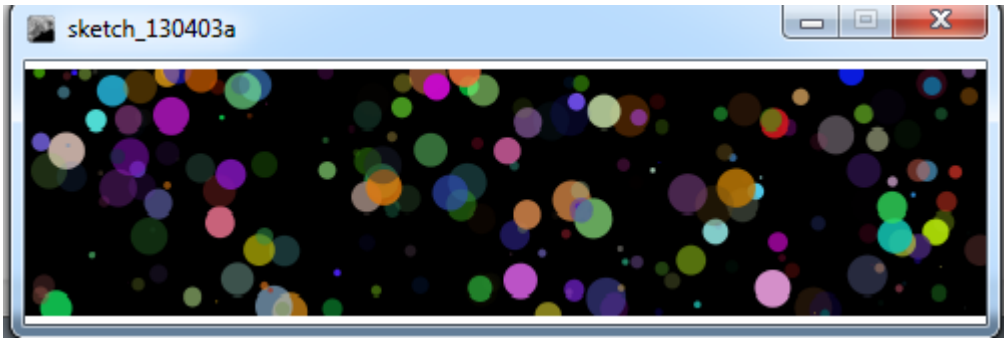


2ο Παράδειγμα

Στο δεύτερο παράδειγμα δημιουργούμε στο παράθυρο της εφαρμογής σφαίρες σε τυχαία θέση, τυχαίου μεγέθους και χρώματος.

```
void setup() {
size(480,120);
background(0);
smooth();
```

```
}
void draw() {
float r  =  random(255);
float g  =  random(255);
float b  =  random(255);
float a  =  random(255);
float diam  =  random(20);
float x  =  random(width);
float y  =  random(height);
noStroke();
fill(r,g,b,a);
ellipse(x,y,diam,diam);
}
```



5.3 Μετρητές χρόνου

Η Processing διαθέτει μία μέθοδο για να μετράει τον χρόνο κάθε προγράμματος ,από την στιγμή που ξεκίνησε μέχρι την στιγμή που σταμάτησε. Η μέθοδος αυτή είναι η millis() η οποία επιστρέφει τον αριθμό των χιλιοστών του δευτερολέπτου από την στιγμή που ξεκίνησε το πρόγραμμα (1000 ms = 1 s). Επιπλέον η μέθοδος δεν μηδενίζει ποτέ όσο εκτελείται το πρόγραμμα επομένως αν αφαιρέσουμε από μια μεταγενέστερη χρονική στιγμή , μια προηγούμενη χρονική στιγμή το αποτέλεσμα θα είναι το χρονικό διάστημα που έχει περάσει. Στο ακόλουθο παράδειγμα θα αλλάζουμε το χρώμα του παραθύρου μας κάθε 5 δευτερόλεπτα δευτερόλεπτα και εμφανίζουμε το αντίστοιχο μήνυμα στην σε κάποια τυχαία θέση.

```
int savedTime = 0;
int totalTime  =  5000;
float x;
float y;
void setup() {
size(300,120);
background(0);
}
void draw() {
int passedTime  =  millis()  -  savedTime;
if (passedTime > totalTime) {
background(random(200));
savedTime  =  millis();
x =  random(width/2);
y =  random(height/2);
text( " 5 seconds have passed! ",x,y );
}
}
```

6 Βιβλιοθήκες

Η κύρια χρήση της γλώσσας Processing είναι η δημιουργία διαδραστικών εφαρμογών και διαδραστικών γραφικών. Ωστόσο την τελευταία δεκαετία η Processing έχει χρησιμοποιηθεί σε πιο περίπλοκα και μεγάλα έργα, όπως για παράδειγμα την δημιουργία μοντέλων για εκτύπωση τριών διαστάσεων, την δημιουργία εικόνων υψηλής ανάλυσης για την παραγωγή κινηματογραφικών εφέ,

την εικονογράφηση μεγάλων εκδόσεων όπως η Nature και οι New York Times. Η Processing έχει την δυνατότητα να παράγει τα παραπάνω έργα και ακόμη περισσότερα, λόγο της χρήσης των βιβλιοθηκών της. Μια βιβλιοθήκη της Processing δεν διαφέρει σε τίποτα από μια βιβλιοθήκη λογισμικού. Δηλαδή οι βιβλιοθήκες της Processing περιέχουν υποβοηθητικό κώδικα και δεδομένα, παρέχοντας, με αυτόν τον τρόπο, υπηρεσίες στην γλώσσα Processing. Η χρήση των βιβλιοθηκών έπαιξε σημαντικό ρόλο στην ανάπτυξη της Processing, γιατί επέτρεπε στους ερευνητές να προσθέτουν νέα χαρακτηριστικά στην γλώσσα γρήγορα.

Αυτήν την στιγμή υπάρχει μια πληθώρα βιβλιοθηκών της γλώσσας Processing.Εκτός από τις βιβλιοθήκες που συμπεριλαμβάνονται μαζί με την γλώσσα Processing(ονομάζονται core libraries) ,υπάρχουν πάνω από 100 βιβλιοθήκες που συνδέονται με την Processing. Το σύνολο των βιβλιοθηκών παρατίθεται στην διεύθυνση http://processing.org/reference/libraries/ .

Για να ενσωματώσουμε μια βιβλιοθήκη στον κώδικα μας, χρησιμοποιώντας από το menu του PDE την επιλογή Sketch → Import Library. Επιλέγοντας μία βιβλιοθήκη προσθέτεται μια γραμμή κώδικα στο πρόγραμμα μας, που μας δείχνει ότι η συγκεκριμένη βιβλιοθήκη θα χρησιμοποιηθεί στο τρέχον πρόγραμμα μας. Για παράδειγμα αν χρησιμοποιήσουμε την βιβλιοθήκη της OpenGL , ο κώδικας που θα προστεθεί στο πρόγραμμα μας είναι ο ακόλουθος:

```
import processing.opengl.*;
```

Προτού χρησιμοποιήσουμε μια βιβλιοθήκη πρέπει πρώτα να την κατεβάσουμε στον υπολογιστή μας και να την τοποθετήσουμε στον φάκελο libraries, που βρίσκεται στο φάκελο που δημιουργεί η γλώσσα για να αποθηκεύει τα προγράμματα που δημιουργούμε . Η διαδρομή για αυτόν τον φάκελο είναι σε περιβάλλον Windows C:\Users\User\Documents\Processing. Αν δεν έχει δημιουργηθεί ο φάκελος libraries, δημιουργήστε τον και αποθηκεύστε την βιβλιοθήκη που έχετε κατεβάσει .

Όπως αναφέρθηκε παραπάνω υπάρχουν πάνω από 100 βιβλιοθήκες της Processing,είναι ξεκάθαρο ότι δεν γίνεται να επεκταθούμε σε όλες .Στα επόμενα κεφάλαια θα δούμε μερικές από τις βασικότερες βιβλιοθήκες τις Processing ,που αφορούν την δημιουργία 3D γραφικών ,την εξαγωγή εικόνων υψηλής ανάλυσης , την διαχείριση video, την διαχείριση ήχου ,και την δημιουργία animations .

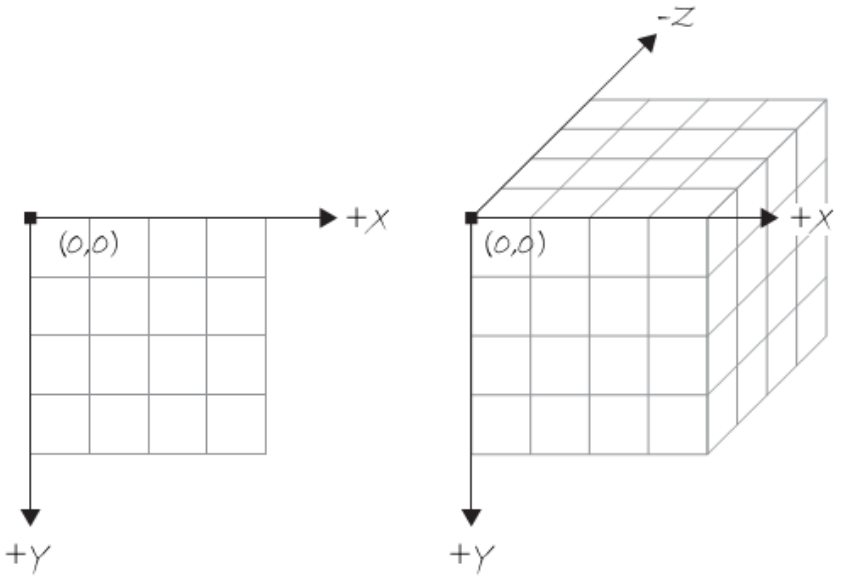
7 Γραφικά 3D

Όπως έχουμε δει παραπάνω για να προσδιορίσουμε την θέση ενός pixel στο παράθυρο μας χρησιμοποιούμε το καρτεσιανό σύστημα δύο διαστάσεων. Σε ένα σύστημα τριών διαστάσεων (όπως για παράδειγμα ο πραγματικός κόσμος), ένας τρίτος άξονας ο οποίος αναφέρεται στο βάθος του κάθε σημείου. Στην Processing ο άξονας Z προσδιορίζει πόσο μπροστά η πίσω τοποθετείται το pixel στο παράθυρο. Στην πραγματικότητα το παράθυρο μας είναι δύο διαστάσεων , και δεν υπάρχουν pixels τοποθετημένα σε βάθος. Με τον άξονα Z μπορούμε να δημιουργήσουμε την ”ψευδαίσθηση” του βάθους .

Στην Processing αν θέλουμε να χρησιμοποιήσουμε ένα σύστημα τριών διαστάσεων υπάρχει ήδη αυτοματοποιημένη διαδικασία της γλώσσας, που δημιουργεί την ψευδαίσθηση του βάθους. Η γλώσσα γνωρίζει για την προοπτική, και επιλέγει τις κατάλληλες συντεταγμένες για τα pixels προκειμένου να δημιουργηθεί το τρισδιάστατο αποτέλεσμα. Με την χρήση συντεταγμένων τριών

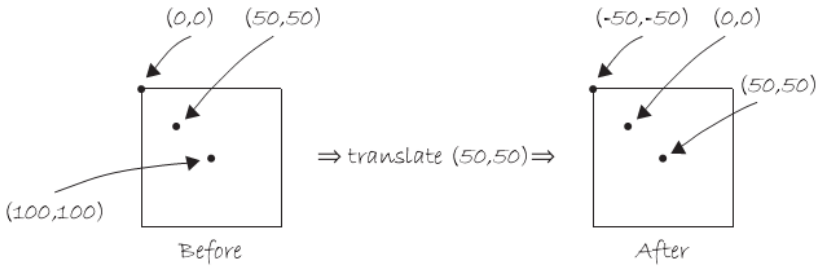
διαστάσεων δεν ελέγχουμε πλήρως την θέση των pixels, όπως γίνεται στην χρήση των συντεταγμένων δυο διαστάσεων. Στην τοποθέτηση ενός pixel στο σύστημα συντεταγμένων των τριών διαστάσεων , μεγάλο ρόλο παίζει ο renderer της Processing .

Για να προσδιορίσουμε ένα σημείο στις τρεις διαστάσεις , διαστάσεις του θα πρέπει να αναφέρονται με την ακόλουθη σειρά X,Y,Z. Το καρτεσιανό σύστημα τριών διαστάσεων μπορούμε να το προσομοιώσουμε με δύο τρόπους , και αυτοί είναι ο " left-handed " ή ο " right-handed. " . Χρησιμοποιούμε αντίστοιχα το αριστερό ή το δεξί χέρι μας για να τους προσομοιώσουμε . Αν χρησιμοποιήσετε το δεξί σας χέρι , τότε ο δείκτης το χεριού σας δείχνει την θετική κατεύθυνση του άξονα Y, ο αντίχειρας σας δείχνει την θετική του άξονα X , και τέλος τα υπόλοιπα δάχτυλα μας θα δείχνουν την θετική κατεύθυνση του άξονα Z. Μπορούμε να χρησιμοποιήσουμε το αριστερό μας χέρι για να κάνουμε την αντίστοιχη προσομοίωση. Στην Processing χρησιμοποιούμε την τον "left-handed " τρόπο όπως φαίνεται στο ακόλουθο σχήμα.



7.1 Σχεδίαση Σχημάτων 3D

Για να σχεδιάσουμε κάποιο από τα γνωστά μας σχήματα ,όπως είναι το παραλληλόγραμμο(rect), σε τρεις διαστάσεις , θα χρειαστούμε την συνάρτηση translate() .Η συνάρτηση translate δεν είναι μια συνάρτηση που χρησιμοποιείται αποκλειστικά για σχήματα τριών διαστάσεων. Η λειτουργία της συνάρτησης αυτής είναι η εξής. Μετακινεί το σημείο αναφοράς (0,0), σε μια νέα θέση σχετική με την προηγούμενη. Στο παρακάτω σχήμα βλέπουμε την χρήση της συνάρτησης translate().



Αφού είδαμε την λειτουργία της συνάρτησης translate() , θα επιστρέψουμε στο αρχικό μας πρόβλημα , πως μπορούμε να προσδιορίσουμε τις συντεταγμένες τριών διαστάσεων με την βοήθεια της συνάρτησης ,translate(). Στο παρακάτω παράδειγμα θα δούμε πως μπορούμε να μεταφέρουμε το σημείο αναφοράς στον άξονα Z και να σχεδιάζουμε ένα παραλληλόγραμμο .

```
float z  =  0;

void setup() {

size(200,200,P3D);

}

void draw() {

background(0);

stroke(255);

fill(100);

translate(width/2,height/2,z);

rectMode(CENTER);

rect(0,0,8,8);

z ++ ;

}
```

Όταν χρησιμοποιούμε συντεταγμένες τύπου (x,y,z) πρέπει να «πούμε» στην Processing ότι πρόκειται για ένα 3D σχέδιο. Αυτό γίνεται προσθέτοντας μια τρίτη παράμετρο(P3D) στην συνάρτηση size(). Παρόλο που δεν είναι εμφανή τα αποτελέσματα στο παραπάνω παράδειγμα , μας ανοίγεται μια νέα προοπτική στην δημιουργία 3D σχεδίων.

7.2 P3D vs. OPENGL

Υπάρχουν δύο τρόποι για να σχεδιάσουμε γραφικά τριών διαστάσεων, και οι δύο απαιτούν να προσθέσουμε μια τρίτη παράμετρο στην συνάρτηση size() ,όπως αναφέραμε παραπάνω, για να αλλάξουμε τον τρόπο που θα σχεδιάζονται τα γραφικά στην οθόνη μας. Πριν δούμε αυτούς τους δύο τρόπους ,θα αναφερθούμε στον εξ ορισμού τρόπο σχεδίασης γραφικών δύο διαστάσεων , που είναι ο " JAVA2D" , ο οποίος χρησιμοποίει τις υπάρχουσες βιβλιοθήκες της Processing για να ορίσει χρώματα, σχήματα και ούτω καθεξής. Δεν χρειάζεται να επεκταθούμε σε λεπτομέρειες για το πώς λειτουργεί ο JAVA2D .Επιστρέφοντας , οι δυο τρόποι για να σχεδιάσουμε γραφικά τριών διαστάσεων είναι ο P3D και ο OpenGL .Παρακάτω βλέπουμε την χρήση της μεθόδου size με δύο και με τρεις παραμέτρους.

```
size(200,200);

size(200,200,P3D);

size(200,200,OPENGL);
```

Ο μεταγλωττιστής P3D είναι ενσωματωμένος με την γλώσσα Processing,πρέπει να σημειωθεί ότι οι συνάρτηση smooth δεν δουλεύει με τον P3D . Ο μεταγλωττιστής της OpenGL , είναι μια βιβλιοθήκη και απαιτεί ενσωμάτωση όπως αναφέραμε πιο πάνω κεφάλαιο. Ο μεταγλωττιστής της OpenGL χρησιμοποιεί τις μέγιστες δυνατότητες που μας προσφέρει το υλικό των ηλεκτρονικών υπολογιστών στις μέρες μας. Πολλές από τις συναρτήσεις που έχουμε δει σε αυτό το βιβλίο είναι λειτουργικές και για τα 3D. Για παράδειγμα οι συναρτήσεις point(),line(), και η vertex() ,δουλεύουν και ως 3D αν προσθέσουμε απλά μια τρίτη παράμετρο συντεταγμένων που αντιπροσωπεύει τον άξονα Z. Παρακάτω θα δούμε δύο παραδείγματα γραφικών 3D ένα με μεταγλωττιστή P3D και ένα με OpenGL .

1^{O} Παράδειγμα

```
size(640, 360, P3D);

background(0);

lights();


noStroke();

pushMatrix();

translate(130, height/2, 0);

rotateY(1.25);

rotateX(-0.4);

box(100);
```

popMatrix();

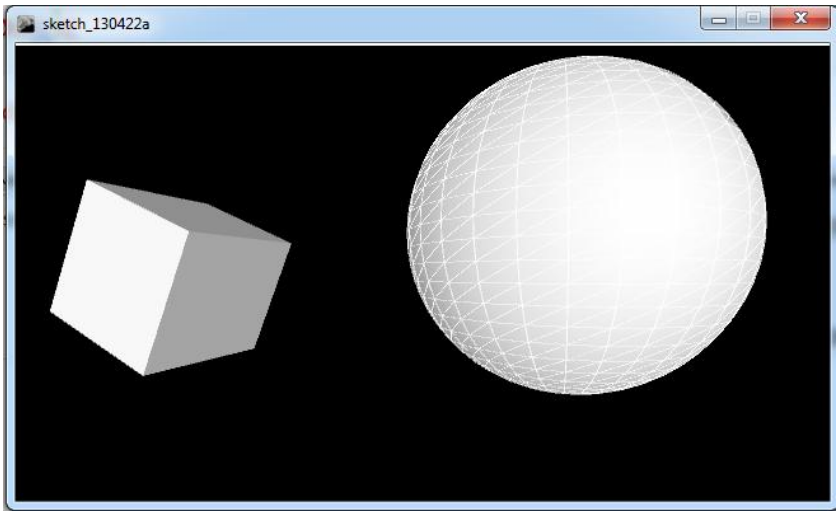
stroke(255);

pushMatrix();

translate(500, height*0.35, -200);

sphere(200);

popMatrix();



Στο παραπάνω παράδειγμα συναντούμε τέσσερις συναρτήσεις για πρώτη φορά , αυτές είναι οι pushMatrix() , popMatrix() ,box() και sphere() .Για τις δύο πρώτες θα μιλήσουμε και παρακάτω αναλυτικά. Οι μέθοδοι box και sphere σχεδιάζουν στην οθόνη 3D σχήματα, μας έναν κύβο και μια σφαίρα αντίστοιχα.

2^{Ο} Παράδειγμα

import processing.opengl.*;

void setup() {

size(440, 220, OPENGL);

noStroke();

fill(255, 190);

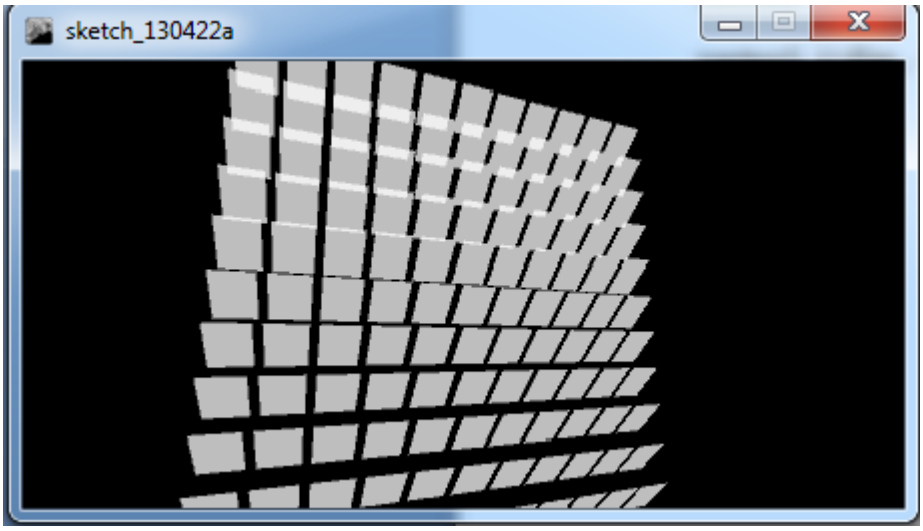
}

void draw() {

background(0);

translate(width/2, height/2, 0);

```
rotateX(mouseX / 200.0);

rotateY(mouseY / 100.0);

int dim = 18;

for (int i = -height/2; i < height/2; i += dim*1.2) {

for (int j = -height/2; j < height/2; j += dim*1.2) {

beginShape();

vertex(i, j, 0);

vertex(i+dim, j, 0);

vertex(i+dim, j+dim, -dim);

vertex(i, j+dim, -dim);

endShape();

}

}

}
```



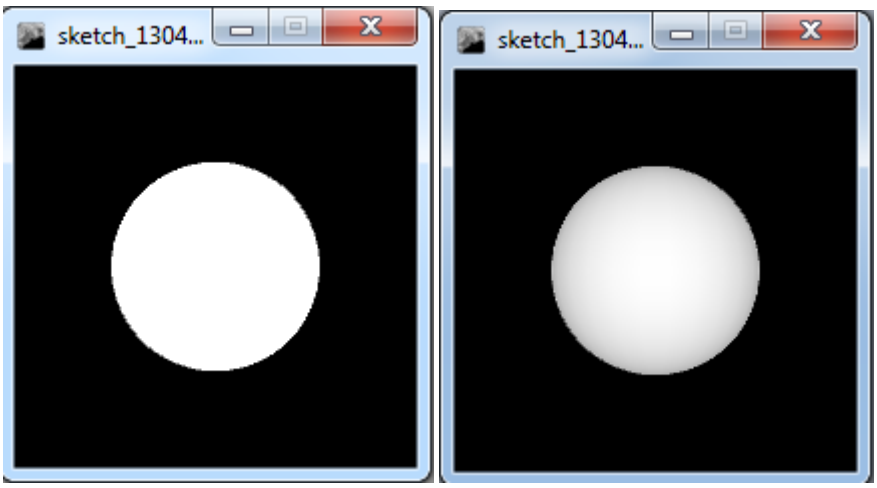
Όταν ξεκινάς να δουλεύεις με 3D σχήματα πολλές νέες συναρτήσεις είναι διαθέσιμες για να χρησιμοποιηθούν. Υπάρχουν δυνατότητες να αλλάξουν την θέση της κάμερας , την προοπτική του φωτισμού και τις υφές των επιφανειών.

7.3 Φωτισμός

Η Processing μας δίνει την δυνατότητα να χειριζόμαστε τον φωτισμό των στοιχείων της οθόνης μας. Φυσικά, όπως και η σχεδίαση στις τρεις διαστάσεις είναι μια ψευδαίσθηση, έτσι και η προσθήκη του

φωτισμού σε ένα 3D σχέδιο της Processing είναι μια προσομοίωση του πραγματικού φωτισμού , με σκοπό τη δημιουργία μιας ποικιλίας των αποτελεσμάτων. Αυτό είναι ιδιαίτερα χρήσιμο δεδομένου ότι ορισμένα αντικείμενα (όπως μια σφαίρα) δεν φαίνεται ότι είναι τρισδιάστατο μέχρι να φωτιστεί κατάλληλα . Αν δεν θέλουμε να μπούμε σε λεπτομέρειες σχετικά με τις μεθόδους που ασχολούνται με τον φωτισμό , μπορούμε να χρησιμοποιήσουμε την εξ ορισμού μέθοδο φωτισμού που είναι η μέθοδος lights() . Στο παρακάτω παράδειγμα θα δούμε την χρήση της συνάρτησης lights() .

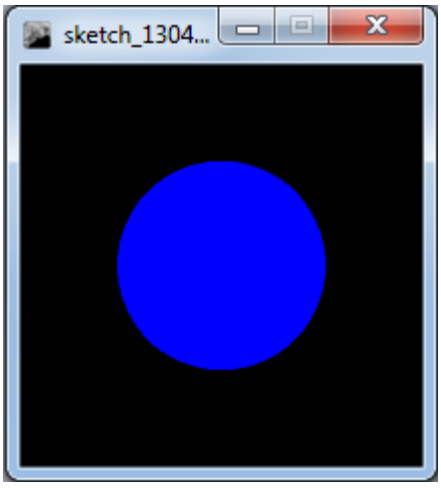
```
import processing.opengl.*;

void setup() {

 size(200, 200, OPENGL);

}

void draw() {

 background(0);

 translate(100, 100, 0);

 if (mousePressed) {

  lights();

 }

 noStroke();

 fill(255);

 sphere(50);

}
```



Για να ορίζουμε τον φωτισμό της αρεσκείας μας υπάρχουν τέσσερις διαφορετικοί τρόποι και οι αντίστοιχες συναρτήσεις τους , που θα τις δούμε αναλυτικά. Η πρώτη συνάρτηση είναι η ambientLight() , η οποία προσθέτει στην οθόνη μας ένα φυσικό φως το οποίο φωτίζει ομοιόμορφα τα αντικείμενα που βρίσκονται σε αυτήν ,από όλες τις πλευρές τους. Αυτό το φυσικό φώς χρησιμοποιείτε σχεδόν πάντα σε συνδυασμό με κάποια άλλη μέθοδο φωτισμού . Για να προσδιοριστεί χρειάζεται ένα χρώμα RGB και

προαιρετικά οι συντεταγμένες(XYZ) της θέσης του φωτός. Αν τροποποιήσουμε τον παραπάνω κώδικα προσθέτοντας την συνάρτηση η ambientLight() θα έχουμε τα εξής αποτελέσματα.

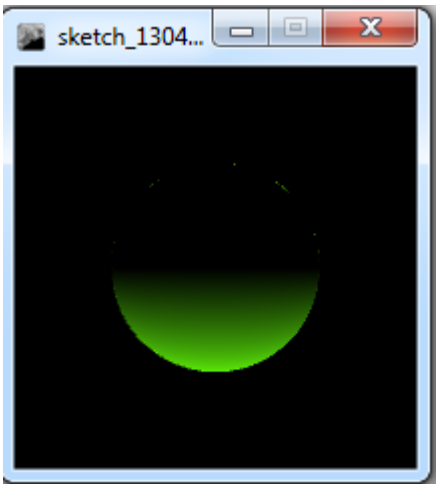
```
import processing.opengl.*;

void setup() {

  size(200, 200, OPENGL);

}

void draw() {

  background(0);

  translate(100, 100, 0);

  ambientLight(0,0,255);

  noStroke();

  fill(255);

  sphere(50);

}
```



Η επόμενη συνάρτηση που θα δούμε είναι η directionalLight() .Η συνάρτηση αυτή δημιουργεί μια πηγή φωτός στην οθόνη μας προερχόμενη από μία συγκεκριμένη κατεύθυνση. Το φώς είναι δυνατότερο όταν χτυπάει στην επιφάνεια μας κάθετα ενώ είναι ασθενέστερο όταν πέφτει σε μια επιφάνεια υπό γωνία. Μετά την πρόσκρουση του φωτός στην επιφάνεια αυτό διαχέεται προς όλες τις κατευθύνσεις. Για να προσδιοριστεί αυτό το φως χρειάζεται ένα χρώμα RGB, καθώς και τις συντεταγμένες ενός διανύσματος που θα προσδιορίζει την κατεύθυνση του φωτός. Αν στο αρχικό μας παράδειγμα θέλουμε να δημιουργήσουμε ένα πράσινο φώς που θα προέρχεται από το κάτω μέρος της οθόνης μας ,πρέπει να προσθέσουμε την εξής.

```
import processing.opengl.*;

void setup() {

  size(200, 200, OPENGL);
```

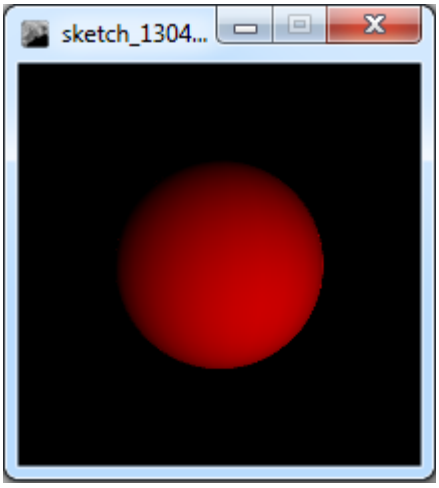
```
}
void draw() {
  background(0);
  translate(100, 100, 0);
directionalLight(0, 255, 0, 0, -1, 0);
  noStroke();
  fill(255);
  sphere(50);
}
```



Η τρίτη συνάρτηση που μπορούμε να χρησιμοποιήσουμε για να πετύχουμε τον φωτισμό της αρεσκείας μας είναι η spotLight() .Η συνάρτηση αυτή είναι παρόμοια με την προηγούμενη, δημιουργεί και αυτή μια πηγή φωτός προερχόμενη από μία κατεύθυνση ,αλλά έχουμε την δυνατότητα να ελέγχουμε το αποτέλεσμα του φωτισμού με μεγαλύτερη ακρίβεια. Ακριβώς όπως πριν, το φως ορίζεται με ένα χρώμα και την κατεύθυνση του. Ωστόσο, απαιτεί επίσης ένα σημείο XYZ για το φως, καθώς και μια γωνία που ελέγχει την εστίαση του φωτός. Μια μικρή τιμή γωνίας θα οδηγήσει σε μια εξαιρετικά εστιασμένη πηγή φωτός ενώ μια μεγαλύτερη γωνία θα έχει ως το αντίθετο αποτέλεσμα . Τέλος, ένα τελευταίο όρισμα καθορίζει τη συγκέντρωση του φωτός, ως προς το κέντρο του κώνου της πηγής του φωτός. Θα δούμε ένα παράδειγμα για την συγκεκριμένη συνάρτηση.

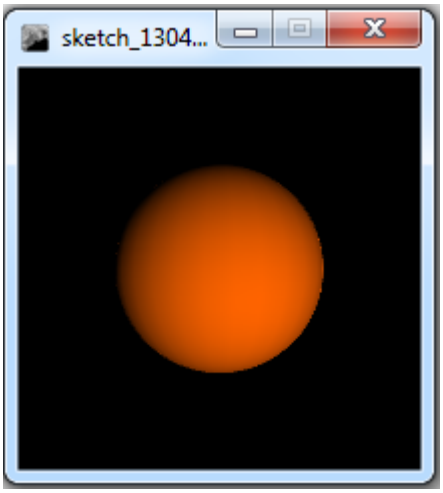
```
import processing.opengl.*;
void setup() {
  size(200, 200, OPENGL);
}
void draw() {
  background(0);
```

```
  translate(100, 100, 0);
spotLight(255, 0, 0, width/2, height/2, 400, 0, 0, -1, PI/4, 4);
  noStroke();
  fill(255);
  sphere(50);
}
```



Τέλος θα δούμε την συνάρτηση pointLight() . Η συγκεκριμένη συνάρτηση δημιουργεί μια πηγή φωτός που μοιάζει με την παραπάνω , η μόνη διαφορά είναι ότι ο κώνος της ακτίνας είναι περιορισμένος στις 180 μοίρες. Για να προσδιορίσουμε την συνάρτηση pointLight() χρειαζόμαστε ένα χρώμα RGB, καθώς και τις συντεταγμένες ενός διανύσματος που θα προσδιορίζει την κατεύθυνση του φωτός.

```
import processing.opengl.*;
void setup() {
  size(200, 200, OPENGL);
}
void draw() {
  background(0);
  translate(100, 100, 0);
pointLight(255, 100, 0, width/2, height/2, 400);
  noStroke();
  fill(255);
  sphere(50);
}
```



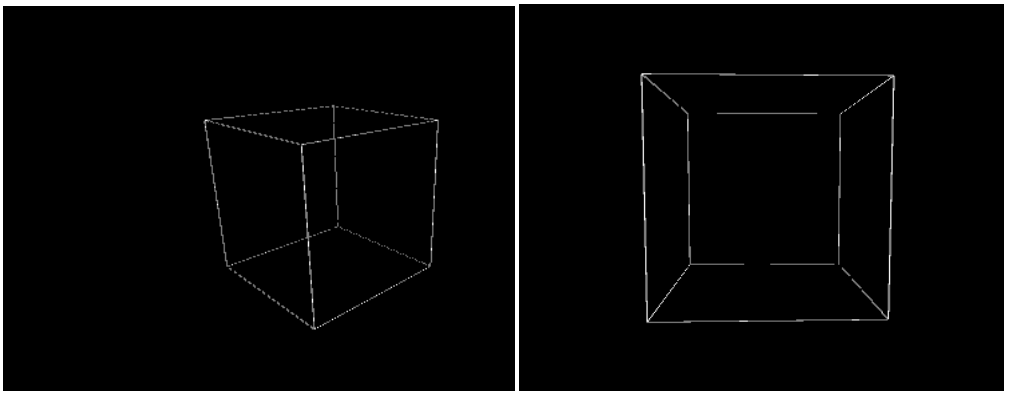
7.4 Η θέση της κάμερας.

Όταν βλέπουμε ένα γραφικό τριών διαστάσεων στο παράθυρο μας μπορούμε να φανταστούμε ότι είναι μια σκηνή από μια κάμερα. Μπορούμε να κάνουμε ζουμ για να φέρουμε πιο κοντά τα αντικείμενα που βλέπουμε , να περιστρέψουμε την κάμερα γύρω από το αντικείμενο μας και να πάρουμε μια σφαιρική του άποψη. Φυσικά, δεν υπάρχει καμία πραγματική κάμερα, αυτό είναι μόνο ένα βολική παρομοίωση για να μας βοηθήσει να καταλάβουμε πώς μπορούμε να διασχίσουμε μια 3D σκηνή. Η προσομοίωση της λειτουργίας της κάμερας μπορεί να γίνει με έξυπνες μετατροπές και χρήση των συναρτησεων translate(), rotate(), και scale() . Παρ 'όλα αυτά, για λόγους ευκολίας, υπάρχει επίσης μια έτοιμή συνάρτηση από την Processing , η camera() ο σκοπός της οποίας είναι επίσης να προσομοιώσει την λειτουργία της κάμερας. Από προεπιλογή η συνάρτηση camera() μας δείχνει το κέντρο της οθόνης μας .Οι παράμετροι που παίρνει η συνάρτηση αυτή είναι αρκετές στο αριθμό , αλλά για την εύκολη κατανόηση της συνάρτησης θα τις χωρίσουμε σε τρείς ομάδες. Οι πρώτες τρεις συντεταγμένες (XYZ) μας δείχνουν την θέση της κάμερας, οι επόμενες τρεις μας δίνουν τις συνταραγμένες (XYZ) το σημείου που εστιάζει η κάμερα, και οι τελευταίες τρεις συντεταγμένες ορίζουν έναν άξονα με κατεύθυνση προς τα επάνω που μας βοήθα να ευθυγραμμίζουμε την κάμερα κάθετα. Θα ακολουθήσουν δύο παραδείγματα χειρισμού της κάμερας, στο πρώτο παράδειγμα θα τροποποιούμε τις συντεταγμένες της θέσης της κάμερας για να δούμε το αντικείμενο από διαφορετική οπτική γωνιά, ενώ στο δεύτερο τροποποιούμε ταυτόχρονα τις συντεταγμένες της θέσης της κάμερας και του κέντρου της οθόνης δημιουργώντας το εφέ που στην κινηματογραφία ονομάζεται panning (είναι η περιστροφή της κάμερας σε οριζόντιο επίπεδο).

1ο Παράδειγμα

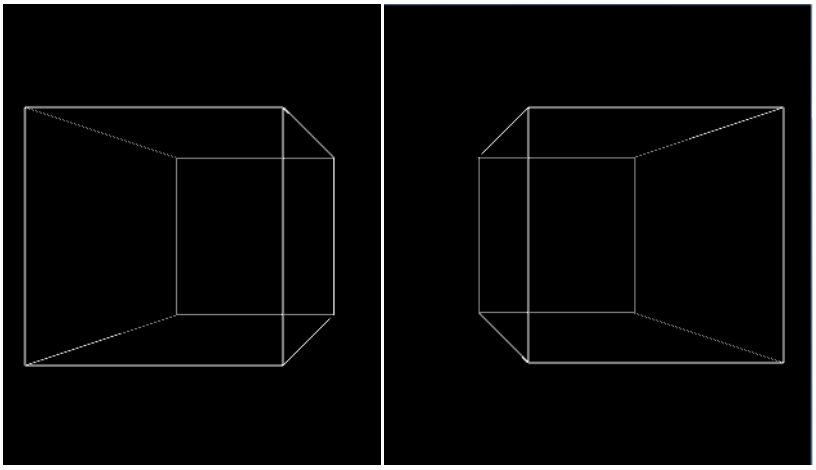
```
import processing.opengl.*;

void setup() {
  size(640, 360, OPENGL);
}

void draw() {
  background(0);
```

```
  camera(mouseX, mouseY/2, (height/2) / tan(PI/6), width/2, height/2, 0, 0, 1, 0);
  translate(width/2, height/2, -100);
  stroke(255);
  noFill();
  box(200);
}
```



2ο Παράδειγμα

```
import processing.opengl.*;
void setup() {
  size(640, 360, OPENGL);
}
void draw() {
  background(0);
  camera(mouseX, height/2, (height/2) / tan(PI/6), mouseX, height/2, 0, 0, 1, 0);
  translate(width/2, height/2, -100);
  stroke(255);
  noFill();
  box(200);
}
```

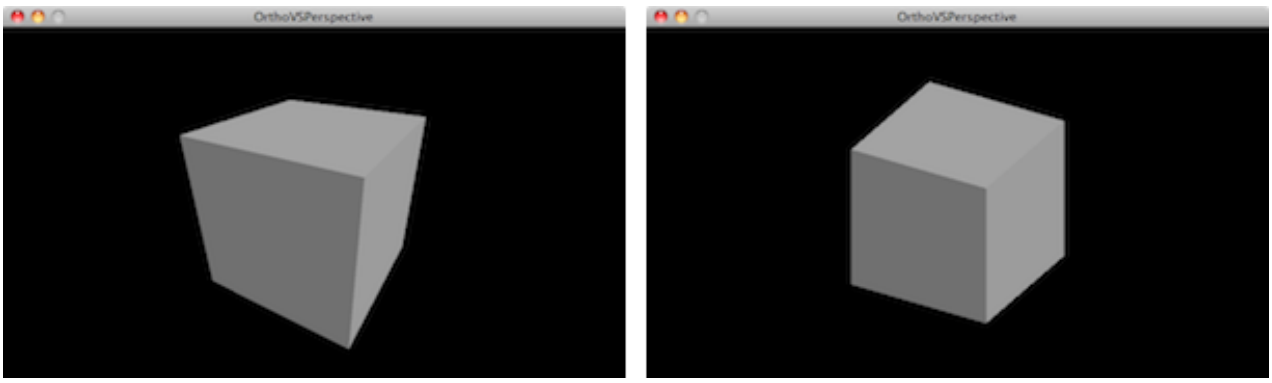


7.5 Προοπτική

Υπάρχουν δύο τρόποι που η Processing χρησιμοποιεί για να δημιουργήσει την προοπτική της απεικόνισης των 3D γραφικών . Ο πρώτος τρόπος ονομάζεται "Perspective" χρησιμοποιεί την προοπτική να παρουσιάζει τα αντικείμενα που είναι μακρύτερα , ως πιο μικρά. Αυτός είναι και ο εξ ορισμού τρόπος που χρησιμοποιεί η Processing. Στις περισσότερες περιπτώσεις, δεν χρειάζεται να καθορίσετε τις παραμέτρους της προοπτικής προβολής, αλλά μπορείτε με την χρήση της συνάρτησης perspective() . Ο άλλος τρόπος προβολής είναι γνωστός ως "orthographic" . Σε αυτή την λειτουργία, όλα τα αντικείμενα της ίδια διάστασης παρουσιάζονται με το ίδιο μέγεθος, ανεξάρτητα από το αν είναι κοντά ή μακριά από την κάμερα. Αυτό είναι συνήθως χρησιμοποιούνται για να επιτευχθεί ένα συγκεκριμένο οπτικό στυλ, όπως αυτό που χρησιμοποιήθηκε στα πρώτα βιντεοπαιχνίδια και ονομαζόταν Q-Bert. Για να ενεργοποιήσετε αυτόν τον τρόπο προβολής, το μόνο που χρειάζεται να κάνετε είναι να καλέσετε την συνάρτηση ortho() .Η συνάρτηση ortho() δεν απαιτεί κανένα όρισμα, αν και θα δείτε ότι υπάρχουν κάποια τα οποία μπορείτε να τα χρησιμοποιήσετε προαιρετικά να ορίσετε ένα επίπεδο αποκοπής. Θα δούμε ένα συγκριτικό παράδειγμα της χρήσης των δύο μεθόδων, οπού με το κλικ του ποντικιού μας μπορούμε να δούμε και τις δύο μεθόδους.

```
import processing.opengl.*;

void setup()  {

  size(640, 360, OPENGL);

  noStroke();

  fill(204);

}

void draw()  {

  background(0);

  lights();

 if(mousePressed) {
```

```
  float fov = PI/3.0;
  float cameraZ = (height/2.0) / tan(fov/2.0);
  perspective(fov, float(width)/float(height), cameraZ/2.0, cameraZ*2.0);
 } else {
  ortho(0, width, 0, height);
 }
 translate(width/2, height/2, 0);
 rotateX(-PI/6);
 rotateY(PI/3);
 box(160);
}
```

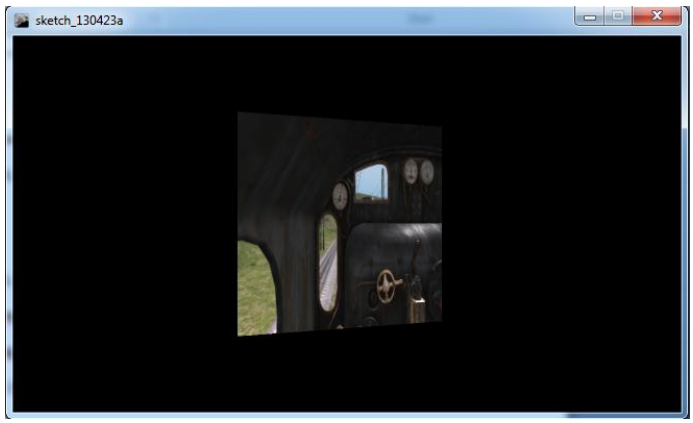


7.6 Υφές επιφανειών

Και στα γραφικά τριών διαστάσεων μπορούμε να φορτώσουμε και να εμφανίσουμε εικόνες ακριβώς όπως κάναμε και στις δύο διαστάσεις. Όλα όσα αναφέραμε για τους μετασχηματισμούς δηλαδή για την χρήση των συναρτήσεων translate(), rotate(), και scale() μπορούν να εφαρμοστούν και στις εικόνες. Εκτός από τον γνωστό τρόπο οι εικόνες μπορούν να χρησιμοποιηθούν και ως υφές επιφανειών αντικειμένων τριών διαστάσεων. Αυτό είναι χρήσιμο όταν θέλουμε να ένα 3D σχήμα να μοιάζει με ένα πραγματικό αντικείμενο. Αν για παράδειγμα χρησιμοποιήσουμε μια εικόνα της γης σε μια σφαίρα θα δημιουργήσουμε μια υδρόγειο. Για να εφαρμόσουμε μια εικόνα ως μια υφή πρέπει πρώτα να ορίσουμε ένα σχήμα με τις μεθόδους beginShape()και endShape() .Την συνάρτηση texture() πρέπει να την καλέσουμε μετά την κλήση της beginShape() και πριν την χρήση της endShape() . Η συνάρτηση texture() δέχεται μόνο ένα όρισμα τύπου PImage. Αφού έχουμε καθορίσει την υφή , θα πρέπει να καθοριστεί στη συνέχεια η αντιστοίχιση της εικόνας στο ίδιο το σχήμα. Αυτό είναι ένα απλό πρόβλημα, όταν το σχήμα είναι ορθογώνιο (τέσσερις γωνίες ενός σχήματος με τις τέσσερις γωνίες μιας εικόνας), αλλά γίνεται όλο και πιο περίπλοκο όταν έχετε πολλές περισσότερες κορυφές σε ένα σχήμα (όπως στο παράδειγμα με την υδρόγειο).

```
PImage img;
```

```
void setup() {
  size(640, 360, OPENGL);
  img = loadImage("train.jpg");
  noStroke();
}
void draw() {
  background(0);
  translate(width / 2, height / 2);
  rotateY(map(mouseX, 0, width, -PI, PI));
  beginShape();
  texture(img);
  vertex(-100, -100, 0, 0, 0);
  vertex(100, -100, 0, img.width, 0);
  vertex(100, 100, 0, img.width, img.height);
  vertex(-100, 100, 0, 0, img.height);
  endShape();
}
```



8 Video

Η βιβλιοθήκη Video της γλώσσας Processing μας επιτρέπει να αναπαράγουμε και να απεικονίσουμε αρχεία βίντεο , να δημιουργήσουμε αρχεία βίντεο από μία κάμερα, να δημιουργήσουμε βίντεο από ένα πρόγραμμα που εκτελείται . Η βιβλιοθήκη Video μπορεί να υποστηρίξει κάμερες με διασύνδεση USB , κάμερες με διασύνδεση IEEE 1394 (Firewire), κάρτες βίντεο , και κάρτες S-video που είναι συνδεδεμένες

στον υπολογιστή μας. Σε αυτό το κεφαλαίο μέσα από παράδειγμα θα εξερευνήσουμε την βιβλιοθήκη Video, τις βασικές κλάσεις της βιβλιοθήκης, που είναι η κλάση Capture και η κλάση Movie, καθώς και τις συναρτήσεις των κλάσεων που αναφέραμε. Πριν ξεκινήσουμε την παρουσίαση της βιβλιοθήκης πρέπει να σιγουρευτούμε ότι έχουμε συνδέσει σωστά την κάμερα στον υπολογιστή μας , να έχουμε εγκαταστήσει το βοηθητικό λογισμικό κα τους drivers της κάμερας, και τέλος και κάνουμε εγκατάσταση τον QuickTime Player (έκδοση 7 η και νεώτερη).

8.1 live Βίντεο

Θα δούμε βήμα βήμα πως μπορούμε αναπαράγουμε σε ένα παράθυρο της Processing ότι ακριβώς κινηματογραφεί η κάμερα μας. Πρώτα από όλα πρέπει να εισάγουμε την βιβλιοθήκη Video στον κώδικα μας , προσθέτοντας την ακόλουθη γραμμή κώδικα

```
import processing.video.*;
```

Στην συνέχεια πρέπει να δημιουργήσουμε ένα αντικείμενο τύπου Capture της βιβλιοθήκης Video .

```
Capture video;
```

Στο επόμενο βήμα πρέπει να καλέσουμε την μέθοδο κατασκευής της κλάσης Capture.Τα ορίσματα της μεθόδου αυτής είναι τέσσερα και είναι τα εξής. Το όρισμα this αναφέρεται σε ένα στιγμιότυπο της κλάσης Capture ,τα δύο επόμενα ορίσματα αναφέρονται στις διαστάσεις του βίντεο που θα αναπαράγουμε (X,Y), και το τελευταίο όρισμα είναι ο ρυθμός frames ανά δευτερόλεπτο του βίντεο που θα αναπαράγουμε.

```
void setup() {

video = new Capture(this,640,480,24);

}
```

Έπειτα πρέπει να καλέσουμε την συνάρτηση start() ώστε να θέλουμε να θέσουμε σε λειτουργία την κάμερα.

```
video.start();
```

Στο επόμενο βήμα θα εξετάσουμε με ποιους τρόπους θα αναγνώσουμε τα καρέ από την κάμερα. Υπάρχουν δύο στρατηγικές για την ανάγνωση των καρέ από την κάμερα. Θα εξετάσουμε εν συντομία και τους δύο τρόπους και αυθαίρετα θα επιλέξουμε τον έναν . Και οι δύο στρατηγικές, όμως, λειτουργούν υπό την ίδια βασική αρχή: θέλουμε μόνο να διαβάσουμε μια εικόνα από την κάμερα, όταν ένα νέο καρέ είναι διαθέσιμο για να το διαβάσουμε. Για να ελέγξουμε αν μια εικόνα είναι διαθέσιμη, χρησιμοποιούμε την συνάρτηση available( ), η οποία επιστρέφει true ή false ανάλογα με το αν είναι διαθέσιμη η όχι το καρέ. Αν υπάρχει καλούμε την συνάρτηση read( ) η οποία διαβάζει το καρέ και το τοποθετεί στην μνήμη. Τοποθετούμε την συνάρτηση read() μέσα στην συνάρτηση draw() , ώστε να ελέγχουμε συνέχεια αν υπάρχει ένα νέο καρέ για διάβασμα .

```
void draw() {

if (video.available()) {

video.read();

}
```

}

Ο άλλος τρόπος που μπορούμε να αναγνώσουμε καρέ από την κάμερα είναι με την χρήση της μεθόδου captureEvent( ), η οποία πρέπει να τοποθετείται έξω από τις μεθόδους draw και setup.

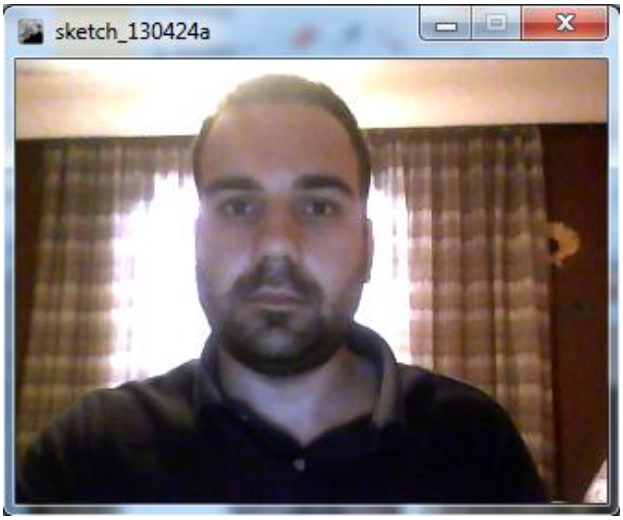
```
void captureEvent(Capture video) {

video.read();

}
```

Το τελευταίο βήμα είναι αναπαραγωγή της το βίντεο στο παράθυρο της Processing . Αυτό είναι, χωρίς αμφιβολία, το πιο εύκολο μέρος. Μπορούμε να σκεφτούμε ένα αντικείμενο Capture ως ένα αντικείμενο PImage που αλλάζει με την πάροδο του χρόνου και, στην πραγματικότητα, ένα αντικείμενο Capture μπορεί να χρησιμοποιηθεί με τον ίδιο τρόπο όπως ένα αντικείμενο PImage.

```
image(video,0,0);
```

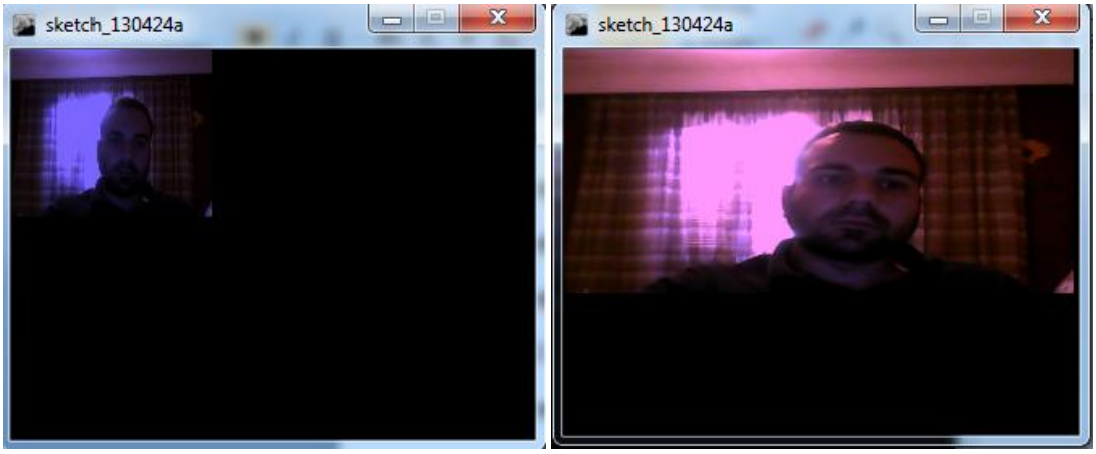
Θα ακολουθήσει ολοκληρωμένο ένα παράδειγμά αναπαραγωγής βίντεο στο παράθυρο της Processing .

```
import processing.video.*;

Capture video;

void setup() {

size(320,240);

video  =  new Capture(this,320,240,24);

video.start();

}

void draw() {

 if (video.available()) {

video.read();

}

image(video,0,0);

}
```



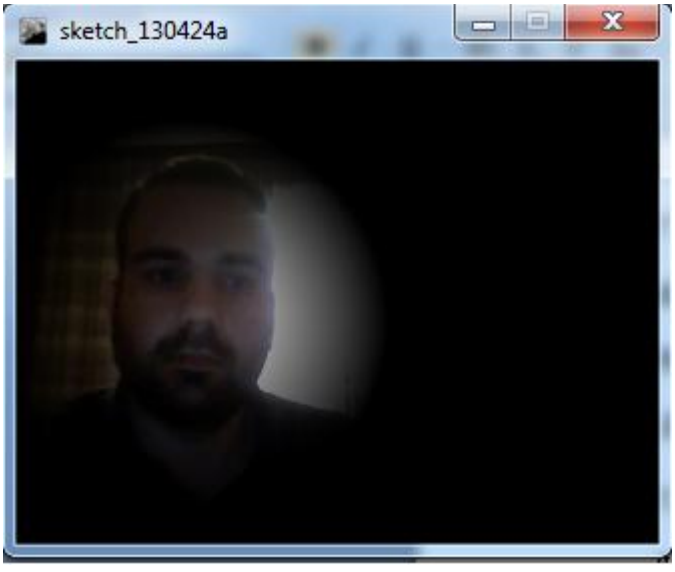
Μπορούμε να κάνουμε ότι κάναμε και με τα αντικείμενα τύπου PImage ,δηλαδή να τα μετακινήσουμε να τα χρωματίσουμε και να τα μετασχηματίσουμε με τις αντίστοιχες μεθόδους .Στο παρακάτω παράδειγμα θα δούμε τον χρωματισμό του βίντεο μας και τις δυναμικές διαστάσεις που μπορούμε να δώσουμε.

```
import processing.video.*;

Capture video;

void setup() {

size(320,240);

video  =  new Capture(this,320,240,24);

video.start();

}

void draw() {

  background(0);

if (video.available()) {

video.read();

}

tint(mouseX,mouseY,255);


image(video,0,0,mouseX,mouseY);

}
```




```
import processing.video.*;
Capture video;
void setup() {
size(320,240);
video  =  new Capture(this,320,240,24);
background(0);
video.start();
}
void draw() {
if (video.available()) {
 video.read();
}
loadPixels();
video.loadPixels();
for (int x  =  0; x<video.width; x ++ ){
for (int y  =  0; y<video.height; y ++ ) {
int loc  =  x  +  y*video.width;
float r,g,b;
r  =  red (video.pixels[loc]);
g  =  green (video.pixels[loc]);
```

```
b = blue (video.pixels[loc]);
float maxdist = 100;// dist(0,0,width,height);
float d = dist(x,y,mouseX,mouseY);
float adjustbrightness = (maxdist-d)/maxdist;
r *= adjustbrightness;
g *= adjustbrightness;
b *= adjustbrightness;
r = constrain(r,0,255);
g = constrain(g,0,255);
b = constrain(b,0,255);
color c = color(r,g,b);
pixels[loc] = c;
}
}
updatePixels();
}
```



8.2 Αναπαραγωγή αρχείου βίντεο

Η αναπαραγωγή ενός αρχείου βίντεο ακολουθεί σε ένα μεγάλο μέρος της την δομή όπως είδαμε παραπάνω .Αρχικά πρέπει να ορίσουμε ένα αντικείμενο τύπου Movie. Αρχικοποιούμε το αντικείμενο Movie . Τα μόνα απαραίτητα ορίσματα είναι το όρισμα this και το όνομα του αρχείου της ταινίας μέσα σε εισαγωγικά. Το αρχείο της ταινίας θα πρέπει να αποθηκευτεί στον φάκελο data του προγράμματος. Ξεκινάμε την αναπαραγωγή της ταινίας , είτε με την βοήθεια της συνάρτησης play() που αναπαράγει την ταινία μας μόνο μια φορά, είτε με την συνάρτηση loop() που αναπαράγει την ταινία συνεχόμενα. Διαβάζουμε τα καρέ του αρχείου του βίντεο με τον ίδιο τρόπο όπως είδαμε παραπάνω. Είτε με την χρήση των συναρτήσεων available() και read() , είτε με την χρήση της συνάρτησης movieEvent(). Τέλος εμφανίζουμε την ταινία στο παράθυρο της Processing.Παρακάτω θα δούμε ένα παράδειγμα ανάγνωσης και αναπαραγωγής ενός αρχείου βίντεο.

```
import processing.video.*;

Movie movie;

void setup() {

size(200,200);

movie = new Movie(this, "test.avi");

movie.loop();

}

void movieEvent(Movie movie) {

movie.read();

}

void draw() {

image(movie,0,0,width,height);

}
```

Θα δούμε ακόμη ένα παράδειγμα όπου θα χρησιμοποιούμε την συνάρτηση jump(), όπου όταν κάνουμε κλικ η ταινία θα μεταβαίνει σε ένα τυχαίο σημείο της.

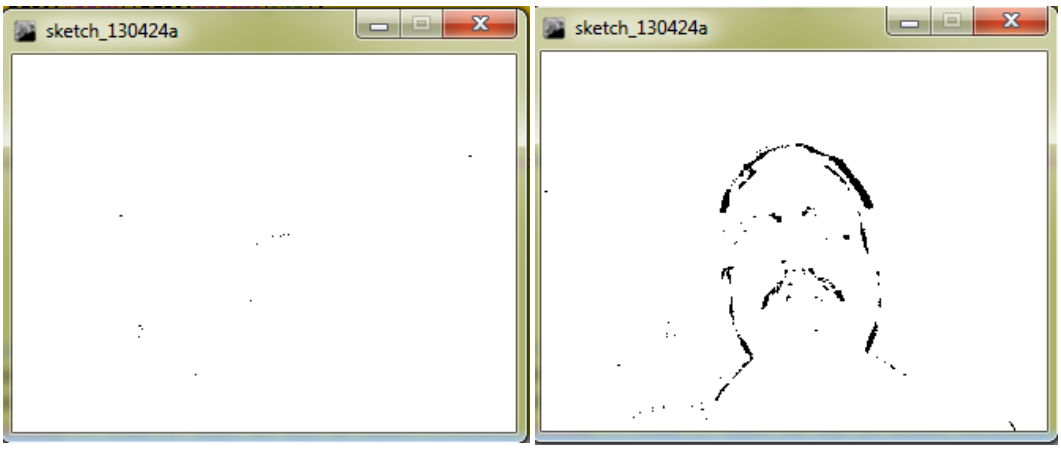
```
import processing.video.*;
Movie myMovie;
void setup() {
  size(200, 200);
  myMovie = new Movie(this, "test.avi");
  myMovie.loop();
}
void draw() {
  if (myMovie.available()) {
    myMovie.read();
  }
  image(myMovie,0,0,width,height);
}
void mousePressed() {
  myMovie.jump(random(myMovie.duration()));
}
```

Τέλος θα δούμε μια εφαρμογή που μπορούμε να δημιουργήσουμε χρησιμοποιώντας την κάμερα μας ως έναν αισθητήρα κίνησης. Η λογική αυτής εφαρμογής είναι η εξής , συγκρίνουμε το προηγούμενο καρέ με το τρέχον καρέ, σαρώνουμε τα pixels των δύο καρέ ένα προς ένα σε όποιο pixel διαπιστώνουμε χρωματική αλλαγή το χρωματίζουμε μαύρο. Η χρωματική αλλαγή του pixel υποδηλώνει κίνηση.σ

```
import processing.video.*;
// Variable for capture device
Capture video;
// Previous Frame
PImage prevFrame;
// How different must a pixel be to be a "motion" pixel
float threshold  =  50;
void setup() {
```

```
size(320,240);
video = new Capture(this, width, height, 24);
// Create an empty image the same size as the video
video.start();
prevFrame = createImage(video.width,video.height,RGB);
}
void draw() {
// Capture video
if (video.available()) {
// Save previous frame for motion detection!!
prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
prevFrame.updatePixels();
video.read();
}
loadPixels();
video.loadPixels();
prevFrame.loadPixels();
// Begin loop to walk through every pixel
for (int x = 0; x < video.width; x ++ ){
for (int y = 0; y < video.height; y ++ ){
int loc = x + y*video.width; // Step 1, what is the 1D pixel location
color current = video.pixels[loc]; // Step 2, what is the current color
color previous = prevFrame.pixels[loc]; // Step 3, what is the previous color
// Step 4, compare colors (previous vs. current)
float r1 = red(current); float g1 = green(current); float b1 = blue(current);
float r2 = red(previous); float g2 = green(previous); float b2 = blue(previous);
float diff = dist(r1,g1,b1,r2,g2,b2);
// Step 5, How different are the colors?
if (diff > threshold) {
```

```
// If motion, display black
pixels[loc]  =  color(0);
} else {
// If not, display white
pixels[loc] = color(255);
}
}
}
updatePixels();
}
```



9 Εξαγωγή εικόνων

Οι animated εικόνες που δημιουργούνται από ένα πρόγραμμα Processing μπορεί να μετατραπεί σε μια ακολουθία αρχείων με την συνάρτηση saveFrame() . Όταν καλέσουμε την συνάρτηση saveFrame () στο τέλος της συνάρτησης draw (), αποθηκεύει μια αριθμημένη ακολουθία εικόνων TIFF της εξόδου του προγράμματος που ονομάζονται screen-0001.tif, screen -0002.tif, και ούτω καθεξής, στο φάκελο data της Processing. Αυτά τα αρχεία μπορούν να εισαχθούν σε ένα πρόγραμμα βίντεο ή να μετατραπούν σε κινούμενα σχέδια και να αποθηκευτούν ως αρχείο ταινίας. Μπορείτε επίσης να καθορίσετε το δικό σας όνομα αρχείου και format αποθήκευσης εικόνας στην συνάρτηση saveFrame() .

```
saveFrame("output-####.png");
```

Χρησιμοποιούμε τον χαρακτήρα # για να δείξουμε που θα τοποθετηθούν αριθμοί στον όνομα του αρχείου μας. Οι χαρακτήρες # αντικαθιστούνται με τον πραγματικό αριθμό του καρέ του προγράμματος. Μπορούμε επίσης να ορίσουμε ένα υποφάκελο για την αποθήκευση των εικόνων μας.

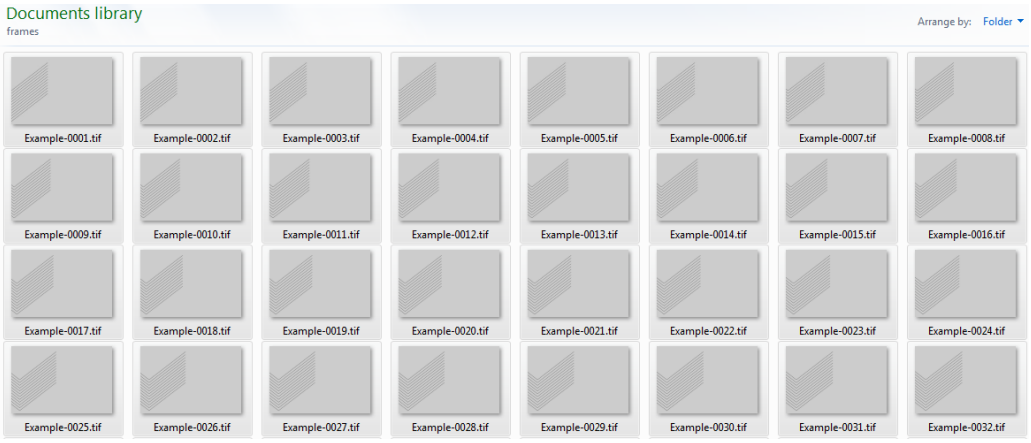
```
saveFrame("frames/output-####.png");
```

9.1 Αποθήκευση εικόνων

Θα δούμε ένα παράδειγμα στο οποίο θα αποθηκεύουμε αρκετές εικόνες ώστε να μπορούμε να δημιουργούμε ένα animation 2 δευτερολέπτων. Το πρόγραμμα θα τρέχει με 30 καρέ ανά δευτερόλεπτο και το πρόγραμμα θα τερματίζει μόλις περάσουν 60 καρέ.

```
smooth();

noFill();

strokeCap(SQUARE);

frameRate(30);

}

void draw() {

background(204);

translate(x, 0);

for (int y = 40; y < 280; y += 20) {

line(-260, y, 0, y + 200);

line(0, y + 200, 260, y);

}

if (frameCount < 60) {

saveFrame("frames/Example-####.tif");

} else {

exit();

}

x += 2.5;

}
```

Στην εικόνα που ακολουθεί βλέπουμε το αποτέλεσμα που δημιουργεί η εκτέλεση του παραπάνω κώδικα. Η Processing αποθηκεύει τις εικόνες βάση της επέκτασης αρχείου που χρησιμοποιούμε.(.png, .jpg, ή .tif) . Οι εικόνες tif αποθηκεύονται σε ασυμπίεστη μορφή , πράγμα το οποίο είναι γρήγορο αλλά μας δημιουργεί μεγάλα αρχεία . Οι εικόνες τύπου png και jpg δημιουργούν μικρότερα αρχεία αλλά χρειάζονται περισσότερο χρόνο για να τα δημιουργήσουν κάνοντας τον κώδικα μας σχετικά αργο.



9.2 Εξαγωγή PDF

Εάν το αποτέλεσμα του κώδικα σας είναι κάποιο διανυσματικό γραφικό, μπορείτε να τα αποθηκεύσετε σε αρχεία PDF για υψηλότερη ανάλυση. Η βιβλιοθήκη PDF καθιστά δυνατή την αποθήκευση των αρχείων PDF απευθείας από το πρόγραμμα μας. Αυτά τα διανυσματικά αρχεία γραφικών μπορούν να προσαρμοστούν σε οποιοδήποτε μέγεθος χωρίς απώλεια ανάλυσης, το οποίο τους καθιστά ιδανικούς για εκτυπώσεις υψηλής ανάλυσης. Θα δούμε μερικά παραδείγματα χρήσης της βιβλιοθήκης PDF.

1ο Παράδειγμα

Σε αυτό το παράδειγμα θα δούμε την αποθήκευση ενός καρέ σε ένα έγγραφο PDF. (Να σημειωθεί ότι δεν δημιουργείτε παράθυρο της Processing κατά την εκτέλεση του κώδικα, αυτό μας βοηθά όταν προσπαθούμε να δημιουργήσουμε μαζικά εικόνες PDF που έχουν πολύ μεγαλύτερη ανάλυση από την ανάλυση της οθόνη μας.)

```
import processing.pdf.*;

void setup() {

  size(400,400,PDF,"test.pdf");

}

void draw() {

  // Draw something good here

  line(0, 0, width/2, height);

   println("Finished.");

  exit();

}
```

2ο Παράδειγμα

Σε αυτό το παράδειγμα θα δούμε την δυνατότητα να αποθηκεύουμε αυτό που βλέπουμε στο παράθυρο μας σε ένα αρχείο PDF,με την βοήθεια των συναρτήσεων beginRecord() και endRecord().Αυτή η διαδικασία είναι πιο αργή σε σχέση με το προηγούμενο παράδειγμα αλλά είναι χρήσιμο όταν θέλουμε να ελέγχουμε οπτικά το αποτέλεσμα.

```
import processing.pdf.*;

void setup() {

 size(400, 400);

 noLoop();

 beginRecord(PDF, "test.pdf");

}

void draw() {

 line(0, 0, width/2, height);

 endRecord();

}
```

3ο Παράδειγμα

Στο παράδειγμα αυτό καταγράφει ότι γίνεται στο παράθυρο της Processing σε ένα αρχείο PDF και τερματίζει το πρόγραμμα όταν πατήσουμε το πλήκτρο q .

```
import processing.pdf.*;

void setup() {

 size(400, 400);

 beginRecord(PDF, "test.pdf");

}

void draw() {

line(mouseX, mouseY, width/2, height/2);

}

void keyPressed() {

 if (key == 'q') {

   endRecord();

   exit();

 }
```

}

4ο Παράδειγμα

Στο τελευταίο παράδειγμα θα δούμε πως μπορούμε να δημιουργήσουμε διανυσματικά γραφικά από 3D γραφικά .Για να το δημιουργήσουμε χρησιμοποιούμε τις συναρτήσεις beginRaw() και endRaw() . Αυτές οι συναρτήσεις αποθηκεύουν τα δεδομένα του σχήματος πριν λίγο πριν αποδοθούν στην οθόνη. Η εγράφη θα γίνεται όταν πατήσουμε το πλήκτρο r.

```
import processing.pdf.*;

boolean record;

void setup() {
  size(500, 500, P3D);
}

void draw() {
  if (record) {
    beginRaw(PDF, "output.pdf");
  }
  background(204);
  translate(width/2, height/2, -200);
  rotateZ(0.2);
  rotateY(mouseX/500.0);
  box(200);


  if (record) {
    endRaw();
    record = false;
  }
}

void keyPressed() {
  if (key == 'r') {
    record = true;
  }
```

```
}
```

10 Ήχος

Η Processing δεν διαθέτει κάποια ενσωματωμένη υποστήριξη για τον ήχο. Η Processing είναι μια γλώσσα προγραμματισμού που το περιβάλλον ανάπτυξης της, είναι στενά συνδεδεμένο με την Java, η γλώσσα αυτή έχει σχεδιαστεί για την εκμάθηση προγραμματισμού σε ένα οπτικό περιβάλλον. Έτσι, αν θέλετε να αναπτύξετε μιας μεγάλης κλίμακας διαδραστική εφαρμογή, που θα είναι κυρίως επικεντρώμένη στον ήχο, θα πρέπει να ρωτήσετε τον εαυτό σας: "Είναι Processing το κατάλληλο περιβάλλον προγραμματισμού ; "Αυτό το κεφάλαιο θα σας βοηθήσει να δώσετε απάντηση στο ερώτημα σας, καθώς θα διερευνάτε τα δυνατότητες και τους περιορισμούς της εργασίας με ήχο στην Processing.

Η ενσωμάτωση της λειτουργία του ήχου στην Processing μπορεί να επιτευχθεί με μια σειρά από διαφορετικούς τρόπους. Επειδή η Processing δεν υποστηρίζει ήχο όπως αναφέραμε παραπάνω, πολλοί προγραμματιστές της Processing επιλέγουν να ενσωματώσουν ήχο στις εφαρμογές τους , μέσω μιας εφαρμογής ενός τρίτου κατασκευαστή, όπως είναι η εφαρμογή PureData (http://www.puredata.org/) ή η εφαρμογή Max / MSP (http://www.cycling74.com/). Η Processing μπορεί να επικοινωνήσει με αυτές τις εφαρμογές μέσω του πρωτοκόλλου OSC, ένα πρωτόκολλο για την επικοινωνία μέσω δικτύου υπολογιστών και συσκευών πολυμέσων.

Αν επισκεφτούμε στην ιστοσελίδα της Processing θα ανακαλύψουμε επίσης έναν κατάλογο από βιβλιοθήκες , που χρησιμοποιώντας αυτές τις βιβλιοθήκες μπορούμε να ενσωματώσουμε τις λειτουργίες του ήχου στις εφαρμογές μας : παίζοντας ηχητικά δείγματα, αναλύοντας είσοδο ήχου από ένα μικρόφωνο, συνθέτοντας ήχου, και κάνοντας αποστολή και λήψη midi πληροφοριών. Αυτό το κεφάλαιο θα επικεντρωθεί σε δύο από αυτά τα στοιχεία: την αναπαραγωγή ήχου και είσοδο ήχου. Για την εξέταση των παραπάνω στοιχείων θα χρησιμοποιήσουμε την βιβλιοθήκη Minim (by Damien Di Fede) http://code.compartmental.net/

10.1 Ένας απλοϊκός τρόπος να αναπαράγουμε ήχο

Πριν δούμε την βιβλιοθήκη Minim, υπάρχει ένας απλός (αλλά εξαιρετικά περιορισμένη) τρόπος για να αναπαράγουμε ένα αρχείο ήχου σε μια εφαρμογή της Processing χωρίς να κάνουμε εγκατάσταση κάποιας βιβλιοθήκης ενός τρίτου κατασκευαστή . Αυτό μπορούμε να το πετύχουμε μέσω της κλάσης Movie της βιβλιοθήκης Video . Η κλάση Movie έχει σχεδιαστεί για να παίξει αρχεία βίντεο, αλλά μπορεί επίσης να χρησιμοποιηθεί για να αναπαράγει ένα αρχείο WAV ή ένα AIFF.

```
import processing.video.*;

Movie movie;

void setup() {

size(200, 200);

movie = new Movie (this, "test.wav" );
```

```
}

void draw() {

background(0);

noLoop();

}

void mousePressed() {

movie.stop();

movie.play();

}
```

10.2 Εισαγωγή στην βιβλιοθήκη Minim

Για αρχίσουμε να χρησιμοποιούμε την βιβλιοθήκη Minim στον κώδικα μας πρέπει πρώτα να δημιουργήσουμε ένα αντικείμενο τύπου Minim , το οποίο στη συνέχεια μπορείτε να το χρησιμοποιήσετε για να φορτώσετε αρχεία ήχου ή να διαχειριστείτε τις εισόδους και εξόδους του ήχου . Στη συνέχεια, πριν τερματίσετε το πρόγραμμα σας, θα πρέπει να κλείσετε οποιαδήποτε κλάση I/O που χρησιμοποιείτε από την βιβλιοθήκη Minim, και στη συνέχεια να σταματήσετε και το αντικείμενο Minim . Οι κλάσεις εισόδου και εξόδου της Minim είναι οι εξής AudioPlayer, AudioSample, AudioSnippet, AudioInput, και AudioOutput. Ένας καλός τρόπος για να εξασφαλιστεί ότι θα τερματίσετε όλες τις κλάσεις του ήχου είναι να προσθέσετε στον κώδικα σας μια συνάρτηση που θα τις τερματίζει. Για να χρησιμοποιήσουμε το αντικείμενο Minim πρέπει να το δηλώσουμε έξω από την συνάρτηση setup(), και στην συνέχεια να αρχικοποιήσουμε τα ορίσματα του αντικειμένου .Θα πρέπει να δημιουργήσουμε το αντικείμενο κατά αυτόν τον τρόπο έτσι ώστε να έχουμε πρόσβασή στον φάκελο data του προγράμματος μας. Θα δούμε ένα απλό παράδειγμα κώδικα όπου θα δημιουργούμε ένα αντικείμενο τύπου Minim.

```
import ddf.minim.*;


Minim minim;


void setup()

{

size(100, 100);


minim = new Minim(this);

}
```

```
void draw()

{ }
```

10.3 Αναπαραγωγή ενός αρχείου ήχου

Υπάρχουν τρεις διαφορετικές κλάσεις για την αναπαραγωγή ενός αρχείου ήχου, η καθεμιά είναι σχεδιασμένη για να αντιμετωπίσει ένα συγκεκριμένο είδος αναπαραγωγής. Οι τρεις αυτές κλάσεις είναι οι : AudioSnippet, AudioSample και AudioPlayer. Για να αρχικοποιήσουμε τα αντικείμενα αυτών των κλάσεων μπορούμε να χρησιμοποιήσουμε τις μεθόδους της Minim που θα δούμε παρακάτω.

loadSnippet(String όνομα αρχείου)

loadSample(String όνομα αρχείου)

loadSample(String όνομα αρχείου, int bufferSize)

loadFile(String όνομα αρχείου)

loadFile(String όνομα αρχείου, int bufferSize)

Το «όνομα αρχείου» μπορεί να είναι ένα αρχείο στο φάκελο data του προγράμματος μας είτε μια απόλυτη διαδρομή προς ένα αρχείο ήχου, ή μια διεύθυνση URL που επιστρέφει ένα αρχείο ήχου (όπως ένα mp3 αρχείο από ένα server ή ακόμα και το URL ενός ραδιοφωνικού διαδικτυακού σταθμού) . Το «bufferSize» ορίζει το πόσο μεγάλο θα είναι το buffer του αντικειμένου μας . Το μέγεθος του buffer του συστήματός για τον ήχο θα είναι ανάλογο με το μέγεθος του «bufferSize» Μια καλή τιμή για το « buffersize» είναι 1024 που είναι και η εξ ορισμού τιμή του. Θα δούμε ένα παράδειγμα κώδικα που φορτώνουμε και αναπαράγουμε ένα αρχείο ήχου, με την βοήθεια της κλάσης AudioPlayer.

```
import ddf.minim.*;

Minim minim;

AudioPlayer in;

void setup()

{

size(512, 200);

minim = new Minim(this);

in = minim.loadFile("song.mp3");

in.play();

}


void draw()
```

```
{

}

void stop()

{

in.close();

minim.stop();

super.stop();

}
```

10.4 Δημιουργία ενός ηχητικού σήματος

Η Minim μας παρέχει τη κλάση AudioOutput για την σύνθεση ήχου και την αναπαραγωγή του. Αυτός είναι ένας ήχος που δημιουργείται κατά την εκτέλεση του προγράμματος σας και στη συνέχεια να τροφοδοτείται στα ηχεία του συστήματος μας. Η AudioOutput μπορεί να χειριστεί τόσο μονοφωνικό όσο και στερεοφωνικό ήχο . Μπορείτε να δηλώσετε ποιον είδος ήχου θα συνθέσετε ,χρησιμοποιώντας την αντίστοιχη μέθοδο της Minim . Αυτές οι μέθοδοι έχουν ως εξής:

```
getLineOut()

getLineOut(int type)

getLineOut(int type, int bufferSize)

getLineOut(int type, int bufferSize, float sampleRate)

getLineOut(int type, int bufferSize, float sampleRate, int bitDepth)
```

Το όρισμα type υποδεικνύει εάν θέλετε ένα στερεοφωνικό ή μονοφωνικό ήχο . Θα πρέπει να χρησιμοποιήσετε ως τιμή είτε την Minim.MONO είτε την Minim.STEREO ως όρισμα. Η εξ ορισμού τιμή για το όρισμα «type» είναι η Minim.STEREO . Το όρισμα buffersize, όπως και παραπάνω, ορίζει το πόσο μεγάλο θα είναι το buffer του συστήματος. Η προεπιλεγμένη τιμή του buffersize είναι 1024. Το όρισμα sampleRate ορίζει το ποσοστό του δείγματος του συνθετικού μας ήχου. Η προεπιλεγμένη τιμή για sampleRate είναι 44100.Τέλος με το όρισμα bitDepth ορίζουμε το βάθος των bit της σύνθεση του ήχου μας . Επί του παρόντος, οι μόνες αποδεκτές τιμές για το bitDepth είναι 8 και 16. Η προεπιλεγμένη τιμή για το bitDepth είναι 16. Θα πρέπει επίσης να σημειωθεί ότι οι παράμετροι της μεθόδου αυτής, που περιγράφουν τα χαρακτηριστικά που θέλετε στην έξοδο του ήχου σας. Στην περίπτωση που το υλικό του υπολογιστή σας δεν υποστηρίζει το συγκεκριμένο σύνολο χαρακτηριστικών ήχου που σας ενδιαφέρουν, τότε αυτή η μέθοδος θα επιστρέψει null. Όταν συμβαίνει αυτό δεν σημαίνει απαραίτητα ότι δεν μπορείτε να πάρετε μια έξοδο σε όλα .

```
import ddf.minim.*;

import ddf.minim.signals.*;
```

```
Minim minim;

AudioOutput out;

SineWave sine;


void setup()

{

size(512, 200);

minim = new Minim(this);

out = minim.getLineOut(Minim.STEREO, 512);

sine = new SineWave(440, 0.5, 44100);

out.addSignal(sine);

}

void draw()

{}

void stop()

{

out.close();

minim.stop();

super.stop();

}
```

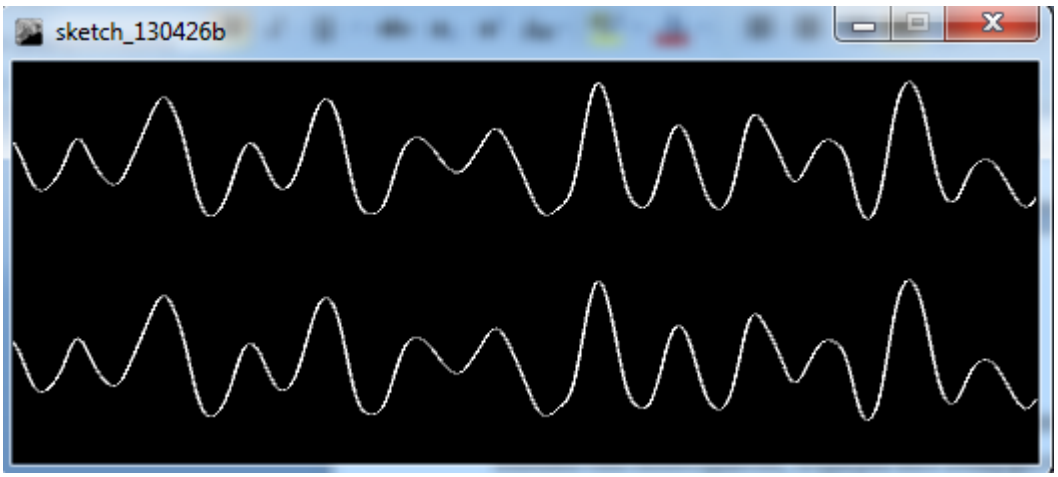
10.5 Διαχείριση μιας εισόδου ήχου

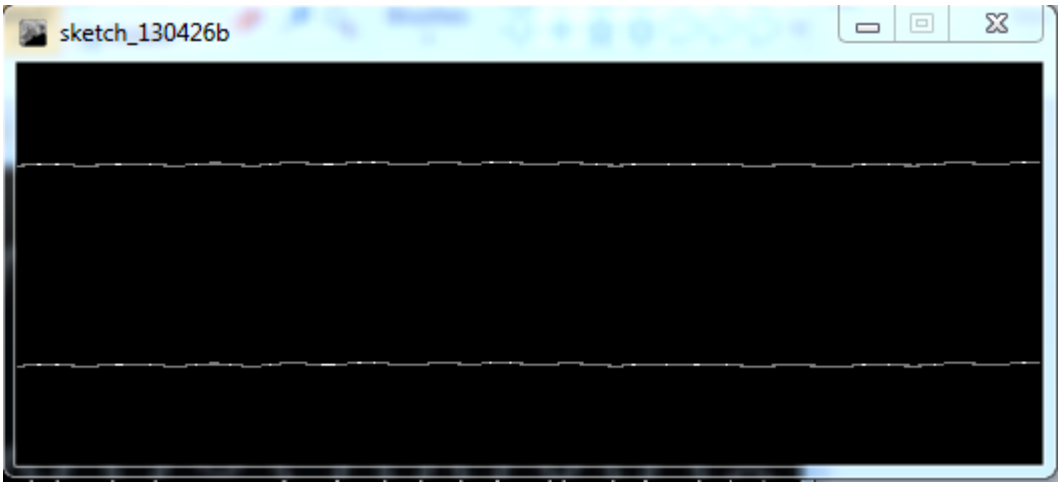
Η βιβλιοθήκη Minim παρέχει την κλάση AudioInput για την παρακολούθηση της τρέχουσας πηγής είσοδο του συστήματος. Σήμερα δεν υπάρχει τρόπος για να ζητήσει ένα συγκεκριμένο τύπο εισόδου, όπως ένα μικρόφωνο ή ένα line-in. Παρ 'όλα αυτά, οι μέθοδοι για την απόκτηση εισόδου της AudioIutput είναι περίπου ίδιες με τις μεθόδους της AudioOutput. Τα ορίσματα έχουν την ίδια συμπεριφορά όπως και στην AudioOutput.

```
getLineIn()

getLineIn(int type)

getLineIn(int type, int bufferSize)
```

```
getLineIn(int type, int bufferSize, float sampleRate)
getLineIn(int type, int bufferSize, float sampleRate, int bitDepth)
import ddf.minim.*;

Minim minim;
AudioInput in;

void setup()
{
  size(512, 200, P3D);

  minim = new Minim(this);
  minim.debugOn();

  in = minim.getLineIn(Minim.STEREO, 512);
}

void draw()
{
  background(0);
  stroke(255);
  for(int i = 0; i < in.bufferSize() - 1; i++)
  {
    line(i, 50 + in.left.get(i)*50, i+1, 50 + in.left.get(i+1)*50);
    line(i, 150 + in.right.get(i)*50, i+1, 150 + in.right.get(i+1)*50);
  }
}
void stop()
{
```

```
  in.close();
  minim.stop();


  super.stop();
}
```





10.6 Αποθήκευση ηχογράφησης στον δίσκο

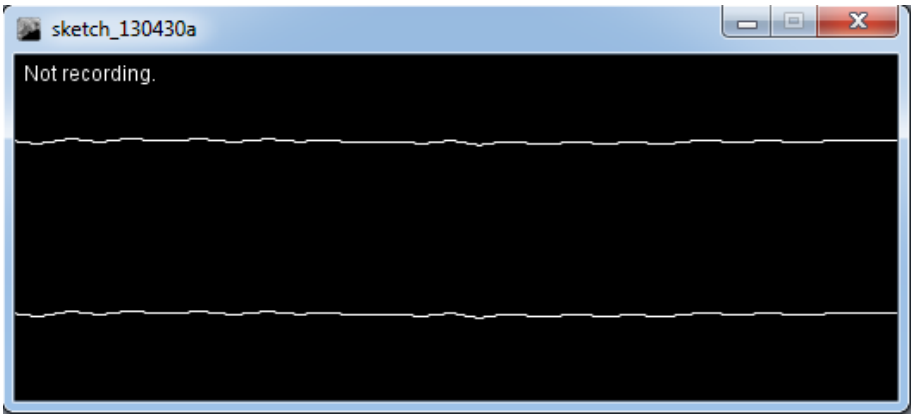
Η βιβλιοθήκη Minim μας παρέχει την κλάση AudioRecorder για την καταγραφή δεδομένων ήχου στον δίσκο μας. Μπορούμε να δημιουργήσουμε ένα αρχείο ήχου με την συνάρτηση createRecorder() η οποία έχει τρία ορίσματα. Το πρώτο όρισμα της συνάρτησης είναι η πηγή εισόδου, το δεύτερο όρισμα είναι το όνομα του αρχείου που θέλουμε να χρησιμοποιήσουμε, πρέπει να προσθέσουμε ότι στο όνομα του αρχείου πρέπει να συμπεριλάβουμε και την επέκταση του αρχείου .Το τρίτο και τελευταίο όρισμα είναι μια μεταβλητή τύπου Boolean η οποία ορίζει αν θα χρησιμοποιηθεί ο buffer στην διαδικασία της ηχογράφησης. Στο παρακάτω παράδειγμα δημιουργούμε ένα αρχείο τύπου wav που περιέχει την ηχογράφηση από το μικρόφωνό μας.

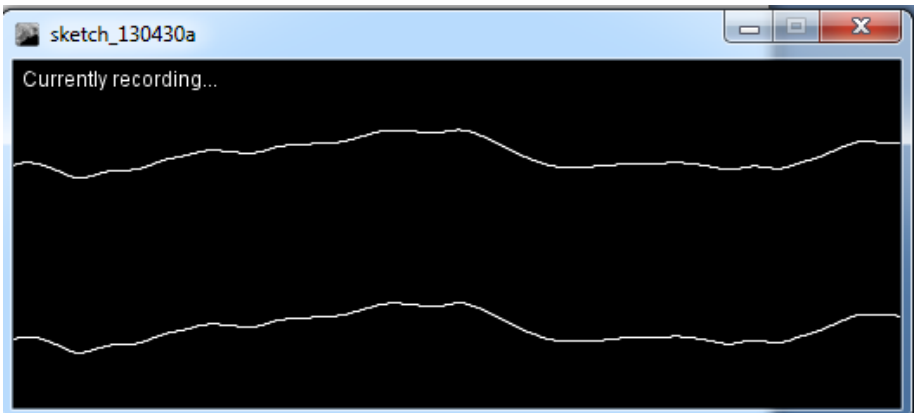
```
import ddf.minim.*;
```

```
Minim minim;
AudioInput in;
AudioRecorder recorder;

void setup()
{
size(512, 200);

minim = new Minim(this);

in = minim.getLineIn(Minim.STEREO, 512);
recorder = minim.createRecorder(in, "myrecording.wav", true);

textFont(createFont("Arial", 12));
}

void draw()
{
background(0);
stroke(255);
for(int i = 0; i < in.left.size()-1; i++)
{
line(i, 50 + in.left.get(i)*50, i+1, 50 + in.left.get(i+1)*50);
line(i, 150 + in.right.get(i)*50, i+1, 150 + in.right.get(i+1)*50);
}

if ( recorder.isRecording() )
{
text("Currently recording...", 5, 15);
```

```
}
else
{
text("Not recording.", 5, 15);
}
}

void keyReleased()
{
if ( key == 'r' )
{

if ( recorder.isRecording() )
{
recorder.endRecord();
}
else
{
recorder.beginRecord();
}
}
if ( key == 's' )
{

recorder.save();
println("Done saving.");
}
}
```

```
void stop()
{

in.close();

minim.stop();

super.stop();
}
```





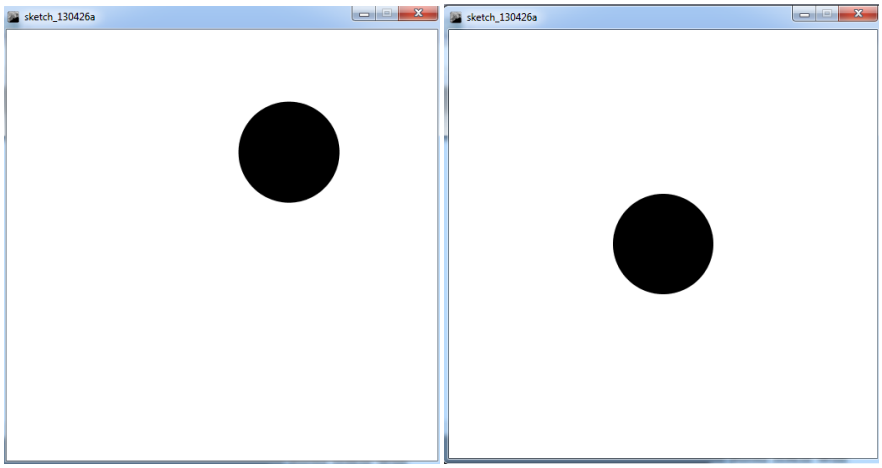
11 Η βιβλιοθήκη Ani

Η βιβλιοθήκη Ani του Benedikt Groß είναι μια βιβλιοθήκη που έχουμε στην διάθεση μας για να δημιουργούμε εύκολα animations και transitions. Ο δικτυακός τόπος της βιβλιοθήκης αυτής είναι ο παρακάτω http://www.looksgood.de/libraries/Ani/ . Η βιβλιοθήκη Ani μέσα από μεθόδους που μας δίνει , επιτρέποντας μας να μετακινούμε αντικείμενα εύκολα στον χώρο . Στο κεφάλαιο αυτό και μέσα από μικρά παραδείγματα θα γνωρίσουμε την βιβλιοθήκη Ani .

11.1 Ένα βασικό παράδειγμα Ani

Σε αυτό το παράδειγμα δημιουργούμε με την βοήθεια της βιβλιοθήκης Ani μια εφαρμογή , στην οποία ένας κύκλος μεταβαίνει στο σημείο όπου έχουμε κάνει κλικ. Θα παρατηρήσετε ότι ο κώδικας μας είναι πολύ πιο σύντομος και κατανοητός. Χρησιμοποιούμε την συνάρτηση to η οποία μετακινεί τον κύκλο μας από την θέση που βρίσκεται στην θέση όπου κάνουμε κλικ. Η τιμή του δεύτερου ορίσματος της συνάρτησης ορίζει τον χρόνο της μετάβασης. Εκτελούμε δύο φορές την συνάρτηση to μία για κάθε συντεταγμένη του κύκλου μας.

```
import de.looksgood.ani.*;

float x = 256;

float y = 256;

void setup() {

  size(512,512);

  smooth();

  noStroke();

  Ani.init(this);

}

void draw() {

  background(255);

  fill(0);

  ellipse(x,y,120,120);

}

void mouseReleased() {


    Ani.to(this, 1.5, "x", mouseX);

    Ani.to(this, 1.5, "y", mouseY);

  }
```



11.2 Callback

Σε αυτό το παράδειγμα θα δούμε πως μπορούμε να δημιουργήσουμε μια εφαρμογή όπου έχουμε ένα κύκλο ο οποίος μεταβαίνει στο σημείο όπου κάναμε κλικ με την βοήθεια της μεθόδου το και η διάμετρος του μεταβάλλεται τυχαία από -20 μέχρι +20 pixel.Στον κώδικα μας θα παρατηρήσουμε έναν από τους αρκετούς δομητές της κλάσης Ani .Θα αναλύσουμε τα ορίσματα του έτσι ώστε να κατανοήσουμε την λειτουργία του animation μας , και πως μπορούμε με μία απλή αλλαγή στον δομητή μας να δώσουμε μια άλλη εικόνα στο animation μας. Ο δομητής μας έχει την εξής μορφή.

diameterAni = new Ani(this, 1.5, "diameter", 150, Ani.EXPO_IN_OUT, "onStart:itsStarted, onEnd:newRandomDiameter");

Το όρισμα this αναφέρεται σε ένα στιγμιότυπο του αντικειμένου της κλάσης Ani,η δεύτερη παράμετρος ορίζει την χρονική διάρκεια του animation ,η τρίτη παράμετρος ορίζει το όνομα του πεδίου που γίνεται το transition, το επόμενο πεδίο ορίζει το τελικό σημείο του transition, η επόμενη μεταβλητή ορίζει τον τρόπο που γίνεται το transition, μπορούμε να βρούμε έναν πλήρη κατάλογο των όλων των δυνατών τρόπων στην ακόλουθή ηλεκτρονική διεύθυνση http://www.looksgood.de/libraries/Ani/Ani_Cheat_Sheet.pdf . Τέλος συναντάμε ένα όρισμα τύπου String το οποίο ονομάζουμε callback στο οποίο μπορούμε να ορίσουμε κάποιες συναρτήσεις να εκτελεστούν είτε όταν το animation αρχίζει είτε όταν τελειώνει είτε τέλος όταν ενημερώνεται. Στο παράδειγμα μας έχουμε ορίσει ότι μόλις ξεκινήσει το animation μας (onStart) θέλουμε να εκτελούμε την συνάρτηση itsStarted() και όταν το animation μας τελειώνει (onEnd) εκτελούμε την συνάρτηση newRandomDiameter(Ani theAni) η οποία υπολογίζει την τυχαία νέα διάμετρο του κύκλου.

```
import de.looksgood.ani.*;

float x = 256;

float y = 256;

int diameter = 50;


Ani diameterAni;
```

```
void setup() {
  size(512,512);
  smooth();
  noStroke();

  Ani.init(this);
 diameterAni = new Ani(this, 1.5, "diameter", 150, Ani.EXPO_IN_OUT, "onStart:itsStarted, onEnd:newRandomDiameter");

  }
void draw() {
  background(255);
  fill(0);
  ellipse(x,y,diameter,diameter);
}
void mouseReleased() {
  Ani.to(this, 1.5, "x", mouseX, Ani.ELASTIC_OUT);
  Ani.to(this, 1.5, "y", mouseY, Ani.ELASTIC_OUT);

  diameterAni.start();
}
void itsStarted() {
  println("diameterAni started");
}

void newRandomDiameter(Ani theAni) {
  float end = theAni.getEnd();
  float newEnd = end + random(-20,20);
```

```
  theAni.setEnd(newEnd);

  println("diameterAni finished. current end: "+end+" -> "+"new end: "+newEnd);

}
```

11.3 Τρόποι transition

Όπως αναφέραμε στο προηγούμενο παράδειγμα υπάρχουν αρκετοί τρόποι για να γίνει ένα transition .Συγκεκριμένα υπάρχουν 26 τρόποι. Στο παράδειγμα που ακολουθεί το οποίο είναι μία εξέλιξη του πρώτου παραδείγματος ,μπορούμε να εναλλάσσουμε τα transitions με το δεξί και το αριστερό πλήκτρο του πληκτρολογίου μας .Έτσι μπορούμε να έχουμε μια γρήγορη εικόνα για το πιο transition ταιριάζει στις ανάγκες μας.

```
import de.looksgood.ani.*;

import de.looksgood.ani.easing.*;

float x = 256;

float y = 256;

int diameter = 50;

Easing[] easings = { Ani.LINEAR, Ani.QUAD_IN, Ani.QUAD_OUT, Ani.QUAD_IN_OUT, Ani.CUBIC_IN, Ani.CUBIC_IN_OUT, Ani.CUBIC_OUT, Ani.QUART_IN, Ani.QUART_OUT, Ani.QUART_IN_OUT, Ani.QUINT_IN, Ani.QUINT_OUT, Ani.QUINT_IN_OUT, Ani.SINE_IN, Ani.SINE_OUT, Ani.SINE_IN_OUT, Ani.CIRC_IN, Ani.CIRC_OUT, Ani.CIRC_IN_OUT, Ani.EXPO_IN, Ani.EXPO_OUT, Ani.EXPO_IN_OUT, Ani.BACK_IN, Ani.BACK_OUT, Ani.BACK_IN_OUT, Ani.BOUNCE_IN, Ani.BOUNCE_OUT, Ani.BOUNCE_IN_OUT, Ani.ELASTIC_IN, Ani.ELASTIC_OUT, Ani.ELASTIC_IN_OUT};

int index = 26;

Easing  currentEasing = easings[index];

void setup() {


  size(512,512);

  smooth();

  noStroke();

 Ani.init(this);

}

void draw() {

  background(255);
```

```
  fill(0);
  ellipse(x,y,diameter,diameter);
}
void mouseReleased() {
  Ani.to(this, 1, "x", mouseX, easings[index]);
  Ani.to(this,1, "y", mouseY, easings[index]);
}
void keyReleased() {
  if (keyCode == LEFT) index--;
  if (keyCode == RIGHT) index++;
  index = constrain(index,0,easings.length-1);
}
```

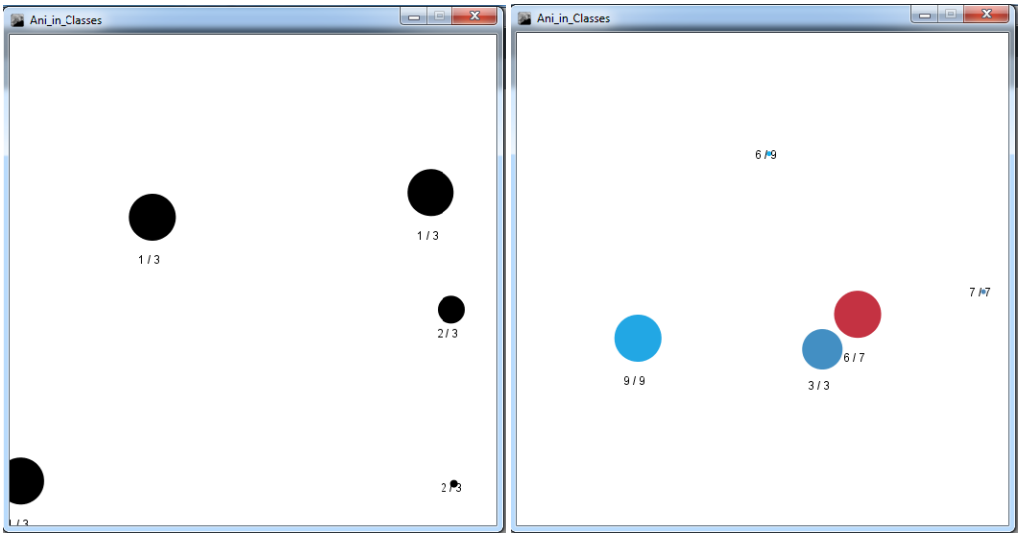
11.4 Η βιβλιοθήκη Ani στις κλάσεις μας

Μπορούμε να ενσωματώσουμε τις λειτουργίες της βιβλιοθήκης Ani στις κλάσεις μας για να τους δώσουμε λειτουργίες animation.Στο συγκεκριμένο παράδειγμα που θα δούμε ,θα δημιουργήσουμε μία κλάση με το όνομα Circle όπου τα αντικείμενα μας θα έχουν λειτουργία animation .Τα αντικείμενα μας θα αλλάζουν χρώμα και θέση σε τυχαίες χρονικές στιγμές με ένα συγκεκριμένο transition και θα εκτελούν για όσο χρόνο μένει εκεί ένα διαφορετικό transition .

```
import de.looksgood.ani.*;
Cirlce[] cirlces = new Cirlce[5];
color from = color(255, 8, 8);
color to = color(8, 187, 255);
void setup() {
  size(512,512);
  smooth();
  noStroke();
  textAlign(CENTER);
  Ani.init(this);
  for(int i=0; i<cirlces.length; i++) {
    cirlces[i] = new Cirlce();
```

```
  }
 }
 void draw() {
  background(255);
 for(int i=0; i<cirlces.length; i++) cirlces[i].draw();
 }
 class Cirlce {
  float x = random(0,width);
  float y = random(0,height);
  int diameter = 5;
  Ani diameterAni;
  color c = color(0);
 Cirlce() {

  diameterAni = new Ani(this, random(1,5), 0.5, "diameter", 50, Ani.EXPO_IN_OUT,
"onEnd:randomize");
  diameterAni.setPlayMode(Ani.YOYO);
  diameterAni.repeat(3);
  }
  void draw() {
   fill(c);
   ellipse(x,y,diameter,diameter);
   fill(0);
   text(diameterAni.getRepeatNumber()+" / "+diameterAni.getRepeatCount(), x, y+diameter);
  }
 void randomize(Ani _ani) {
   c = lerpColor(from, to, random(1));
  int newCount = 1+2*round(random(4));
   diameterAni.repeat(newCount);
```

```
diameterAni.start();
Ani.to(this, 1.5, "x", random(0,width), Ani.EXPO_IN_OUT);
 Ani.to(this, 1.5, "y", random(0,height), Ani.EXPO_IN_OUT);
  }
}
```



12 Βιβλιογραφία

1) Reas Casey & 2010 Getting Started with Processing O'Reilly Media
2) Fry Ben 2008 Visualizing Data O'Reilly Media
3) Shiffman Daniel 2008 Learning Processing A Beginner's Guide to Programming Images, Animation, and Interaction Elsevier
4) http://www.processing.org/
5) http://code.compartmental.net/tools/minim/
6) http://www.looksgood.de/libraries/Ani/